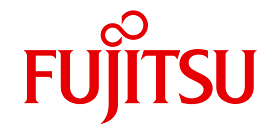
Benutzerhandbuch - Deutsch



openUTM-Client V6.1 für Trägersystem UPIC

Client-Server-Kommunikation mit openUTM

Ausgabe Dezember 2011

Kritik… Anregungen… Korrekturen…

Die Redaktion ist interessiert an Ihren Kommentaren zu diesem Handbuch. Ihre Rückmeldungen helfen uns, die Dokumentation zu optimieren und auf Ihre Wünsche und Bedürfnisse abzustimmen.

Sie können uns Ihre Kommentare per E-Mail an manuals@ts.fujitsu.com senden.

Zertifizierte Dokumentation nach DIN EN ISO 9001:2008

Um eine gleichbleibend hohe Qualität und Anwenderfreundlichkeit zu gewährleisten, wurde diese Dokumentation nach den Vorgaben eines Qualitätsmanagementsystems erstellt, welches die Forderungen der DIN EN ISO 9001:2008 erfüllt.

cognitas. Gesellschaft für Technik-Dokumentation mbH
www.cognitas.de

Copyright und Handelsmarken

Inhalt

1 Einleitung 11

1.1 Kurzbeschreibung des Produkts openUTM-Client 13

1.2 Zielgruppe und Konzept des Handbuchs 14

1.3 Wegweiser durch die Dokumentation zu openUTM 15
1.3.1 openUTM-Dokumentation 15
1.3.2 Dokumentation zum openSEAS-Produktumfeld 19
1.3.3 Readme-Dateien 20

1.4 Änderungen gegenüber dem Vorgängerhandbuch 21

1.5 Darstellungsmittel 22

2 Anwendungsbereich 25

2.1 Das Konzept von openUTM-Client 26

2.2 Client-Server-Kommunikation mit openUTM 28

2.3 UPIC-Local, UPIC-Remote und Multithreading 29

2.4 Unterstützung von UTM-Cluster-Anwendungen 32

3 C++ Klasse CUpic 33

3.1 Einleitung 33
3.1.1 Konfiguration mittels Helper Classes CUpicLocAddr und CUpicRemAddr 33
3.1.2 Konfiguration mit einer Side Information Datei (upicfile) 34
3.1.3 Die CUpic Klasse auf threadfähigen Systemen 34

3.2 Helper Classes 35
3.2.1 CUpicLocAddr 35
3.2.1.1 Konstruktoren 35
3.2.1.2 Member Functions 36

3.2.2 CUpicRemAddr . . . 37
3.2.2.1 Konstruktoren . . . 37
3.2.2.2 Member Functions . . . 38
3.2.3 CUpic Security . . . 39

3.3 ClassCUpic . . . 40
3.3.1 Konstruktoren . . . 40
3.3.2 Property Handlers . . . 40
3.3.3 Funktionsaufrufe . . . 42
3.3.4 Public Diagnosefunktion . . . 46

3.4 Beispiel . . . 47

4 CPI-C-Schnittstelle . . . 49

4.1 CPI-C-Begriffe . . . 50

4.2 Allgemeiner Aufbau einer CPI-C-Anwendung . . . 55

4.3 Austausch von Nachrichten mit einem UTM-Service . . . 56
4.3.1 Nachricht senden und UTM-Service starten . . . 57
4.3.2 Nachricht empfangen, blockierender und nicht-blockierender Receive . . . 59
4.3.3 Formate senden und empfangen . . . 62
4.3.4 UTM-Funktionstasten . . . 66
4.3.5 Cursor-Position . . . 68
4.3.6 Code-Konvertierung . . . 68
4.3.7 Benutzerdefinierte Code-Konvertierung für Windows-Systeme . . . 70

4.4 Kommunikation mit dem openUTM-Server . . . 71
4.4.1 Kommunikation mit einem Einschritt-UTM-Vorgang . . . 72
4.4.2 Kommunikation mit einem Mehrschritt-UTM-Vorgang . . . 75
4.4.3 Kommunikation mit einem Mehrschritt-UTM-Vorgang unter Nutzung von verteilter Transaktionsverarbeitung . . . 76
4.4.4 Transaktionsstatus abfragen . . . 77

4.5 Benutzerkonzept, Security und Wiederanlauf . . . 77
4.5.1 Benutzerkonzept . . . 77
4.5.2 Security-Funktionen . . . 78
4.5.3 Wiederanlauf . . . 81

4.6 Verschlüsselung . . . 84

4.7 Multiple Conversations . . . 89

4.8 DEFAULT-Server und DEFAULT-Name eines Client . . . 94
4.8.1 Mehrfachanmeldungen bei derselben UTM-Anwendung mit demselben Namen . . . 95

4.9 CPI-C-Aufrufe bei UPIC 96
Übersicht 96
Allocate - Conversation einrichten 100
Convert_Incoming - Konvertieren vom Code des Senders in lokalen Code 103
Convert_Outgoing - Konvertieren von lokalem Code in den Code des Empfängers . . 104
Deallocate - Conversation beenden 105
Deferred_Deallocate - Conversation nach Transaktionsende beenden 107
Disable_UTM_UPIC - Vom Trägersystem UPIC abmelden 109
Enable_UTM_UPIC - Beim Trägersystem UPIC anmelden 111
Extract_Client_Context - Client-Kontext abfragen 115
Extract_Communication_Protocol – Zustand der Conversation Characteristic PROTOCOL abfragen 118
Extract_Conversation_Encryption_Level - Verschlüsselungsebene abfragen 120
Extract_Conversation_State - Zustand der Conversation abfragen 123
Extract_Convertion – Wert der Conversation Characteristic CHARACTER_CONVERTION abfragen 125
Extract_Cursor_Offset - Offset der Cursor-Position abfragen 127
Extract_Partner_LU_Name - partner_LU_Name abfragen 129
Extract_Secondary_Information - Erweiterte Information abfragen 131
Extract_Secondary_Return_Code - Erweiterten Returncode abfragen 134
Extract_Shutdown_State - Shutdown-Status des Servers abfragen 139
Extract_Shutdown_Time - Shutdown-Time des Servers abfragen 141
Extract_Transaction_State - Vorgangs- und Transaktionsstatus des Servers abfragen 144
Initialize_Conversation - Conversation Characteristics initialisieren 147
Prepare_To_Receive - Vom Sende- in den Empfangsstatus wechseln 151
Receive - Daten von einem UTM-Service empfangen 154
Receive_Mapped_Data - Daten und Formatkennzeichen von einem UTM-Service empfangen 166
Send_Data - Daten an einen UTM-Service senden 178
Send_Mapped_Data - Daten und Formatkennzeichen senden 181
Set_Allocate_Timer - Timer für den Allocate setzen 184
Set_Client_Context - Client-Kontext setzen 186
Set_Communication_Protocol – Setzen der Conversation Characteristic COMMUNICATION_PROTOCOL 189
Set_Conversation_Encryption_Level - Verschlüsselungsebene setzen 191
Set_Conversation_Security_New_Password - neues Passwort setzen 195
Set_Conversation_Security_Password - Passwort setzen 198
Set_Conversation_Security_Type - Security-Typ setzen 201
Set_Conversation_Security_User_ID - UTM-Benutzerkennung setzen 203
Set_Convertion – Setzen der Conversation Characteristic CHARACTER_CONVERTION 206
Set_Deallocate_Type - Characteristic deallocate_type setzen 208
Set_Function_Key - UTM-Funktionstaste setzen 210
Set_Partner_Host_Name - Hostname der Partner-Anwendung setzen 213

Set_Partner_IP_Address - IP-Adresse der Partner-Anwendung setzen 215
Set_Partner_LU_Name - Setzen der Conversation Characteristics partner_LU_name 218
Set_Partner_Port - TCP/IP-Port der Partner-Anwendung setzen 221
Set_Partner_Tsel - T-SEL der Partner-Anwendung setzen 223
Set_Partner_Tsel_Format - T-SEL-Format der Partner-Anwendung setzen 225
Set_Receive_Timer - Timer für den blockierenden Receive setzen 228
Set_Receive_Type - Empfangsmodus (receive_type) setzen 231
Set_Sync_Level - Synchronisationsstufe (sync_level) setzen 234
Set_TP_Name - TP-Name setzen 236
Specify_Local_Port - TCP/IP-Port der lokalen Anwendung setzen 238
Specify_Local_Tsel - T-SEL der lokalen Anwendung setzen 240
Specify_Local_Tsel_Format - TSEL-Format der lokalen Anwendung setzen 242
Specify_Secondary_Return_Code – Eigenschaften des erweiterten Returncode setzen 244

4.10 COBOL-Schnittstelle 246

5 XATMI-Schnittstelle 249

5.1 Client-Server-Verbund 251
5.1.1 Default-Server 252
5.1.2 Wiederanlauf 252

5.2 Kommunikationsmodelle 253

5.3 Typisierte Puffer 256

5.4 Programmschnittstelle 259
5.4.1 XATMI-Funktionen für Clients 259
5.4.2 Aufrufe für den Anschluss an das Trägersystem 260
tpinit - Client initialisieren 261
tpterm - Client abmelden 263
5.4.3 Transaktionssteuerung 264
5.4.4 Mischbetrieb 264
5.4.5 Administrationsschnittstelle 264
5.4.6 Include-Dateien und COPY-Elemente 265
5.4.7 Ereignisse und Fehlerbehandlung 266
5.4.8 Typisierte Puffer erstellen 267
5.4.9 Characteristics von XATMI in UPIC 269

5.5 Konfigurieren 270
5.5.1 Local Configuration File erzeugen 270
5.5.2 Das Tool xatmigen bzw. xtgen32 275

5.5.3 Trägersystem und UTM-Partner konfigurieren 278
5.5.3.1 UPIC konfigurieren 278
5.5.3.2 Initialisierungsparameter und UTM-Generierung 279

5.6 Einsatz von XATMI-Anwendungen 282
5.6.1 Binden und Starten eines XATMI-Programms 282
5.6.1.1 Binden eines XATMI-Programms unter Windows-Systemen 282
5.6.1.2 Binden eines XATMI-Programms auf Unix-Systemen 282
5.6.1.3 Binden eines XATMI-Programms unter BS2000/OSD 283
5.6.1.4 Starten 283
5.6.2 Umgebungsvariablen auf Windows- und Unix-Systemen setzen 283
5.6.3 Jobvariablen setzen unter BS2000/OSD 285
5.6.4 Trace 286

5.7 Meldungen des Tools xatmigen 287

6 Konfigurieren 291

6.1 Konfigurieren ohne upicfile 292
6.1.1 Konfiguration UPIC-L 293
6.1.2 Konfiguration UPIC-R 294
6.1.3 Konfiguration mit TNS-Einträgen 296
6.1.4 Konfiguration mit BCMAP-Einträgen 296

6.2 Die Side Information-Datei (upicfile) 297
6.2.1 Side Information für stand-alone UTM-Anwendungen 298
6.2.2 Side Information für UTM-Cluster-Anwendungen 305
6.2.3 Side Information für die lokale Anwendung 312

6.3 Abstimmung mit der Partnerkonfiguration 315

7 Einsatz von CPI-C-Anwendungen 319

7.1 Ablaufumgebung, Binden, Starten 319
7.1.1 Einsatz in Windows-Systemen 321
7.1.1.1 Übersetzen, Binden, Starten 321
7.1.1.2 Ablaufumgebung, Umgebungsvariablen 322
7.1.1.3 Besonderheiten beim Einsatz von UPIC-Local auf Windows-Systemen 324
7.1.2 Einsatz in Unix-Systemen 327
7.1.2.1 Übersetzen, Binden, Starten 327
7.1.2.2 Ablaufumgebung, Umgebungsvariablen 328
7.1.2.3 Besonderheiten beim Einsatz von UPIC-Local auf Unix-Systemen 329
7.1.3 Einsatz unter BS2000/OSD 330

7.2 Behandlung von CPI-C-Partnern durch openUTM 332
7.3 Verhalten im Fehlerfall 333
7.4 Diagnose 337
7.4.1 UPIC-Logging-Datei 337
7.4.2 UPIC-Trace 338
7.4.3 PCMX-Diagnose (Windows-Systeme) 344

8 Beispiele 345

8.1 Programmbeispiele für Windows-Systeme 345
8.1.1 uptac 346
8.1.2 utp32 347
8.1.3 tpcall 347
8.1.4 upic-cob 347
8.1.5 UpicSimpleClient 348
8.2 Generierung UPIC auf Windows-System <-> openUTM auf BS2000/OSD 349
8.2.1 Generierung auf dem Windows-System 349
8.2.2 Generierung auf dem BS2000-Rechner 350
8.3 Generierung UPIC auf Windows-System <-> openUTM auf Unix-System 351
8.3.1 Generierung auf dem Windows-System 351
8.3.2 Generierung auf dem Unix-System 352

9 Anhang 353

9.1 Unterschiede zur X/Open-Schnittstelle CPI-C 353
9.2 Zeichensätze 356
9.3 Zustandstabelle 359

Fachwörter 367

Abkürzungen 409

Literatur 415

Stichwörter 423

1 Einleitung

Moderne unternehmensweite IT-Umgebungen unterliegen zahlreichen Herausforderungen von zunehmender Brisanz. Dies wird verursacht durch

- heterogene Systemlandschaften
- unterschiedliche HW-Plattformen
- unterschiedliche Netze und Netzzugriffe (TCP/IP, SNA, HTTP)
- Verflechtung der Anwendungen mit den Unternehmen

Dadurch entwickeln sich Problemfelder, sei es bei Fusionen, durch Kooperationen oder auch nur durch Rationalisierungsmaßnahmen. Die Unternehmen fordern flexible und skalierbare Anwendungen, gleichzeitig soll die Transaktionssicherheit für Prozesse und Daten gewährleistet bleiben, obwohl die Geschäftsprozesse immer komplexer werden. Die wachsende Globalisierung geht selbstverständlich davon aus, dass Anwendungen im 7x24-Stunden-Betrieb laufen und hochverfügbar sind, um beispielsweise Internetzugriffe auf bestehende Anwendungen über Zeitzonen hinweg zu ermöglichen.

Die transaktionsorientierte Middleware-Plattform openUTM bietet eine Ablaufumgebung, die all diesen Anforderungen moderner unternehmenskritischer Anwendungen gewachsen ist, denn openUTM verbindet alle Standards und Vorteile von transaktionsorientierten Middleware-Plattformen und Message Queuing Systemen:

- Konsistenz der Daten und der Verarbeitung
- Hohe Verfügbarkeit der Anwendungen (nicht nur der Hardware)
- Hohen Durchsatz auch bei großen Benutzerzahlen, d.h. höchste Skalierbarkeit
- Flexibilität bezüglich Änderungen und Anpassungen des IT-Systems

Eine UTM-Anwendung kann auf einem einzelnen Rechner als stand-alone UTM-Anwendung und auf mehreren Rechnern als UTM-Cluster-Anwendung betrieben werden.

openUTM ist Teil des umfassenden Angebots von **openSEAS**. Gemeinsam mit der Oracle Fusion Middleware bietet openSEAS die komplette Funktionalität für Anwendungsinnovation und moderne Anwendungsentwicklung. Im Rahmen des Produktangebots **openSEAS** nutzen innovative Produkte die ausgereifte Technologie von openUTM:

- BeanConnect ist ein Adapter gemäß der Java Connector Architecture (JCA) von Oracle/Sun und bietet den standardisierten Anschluss von UTM-Anwendungen an J2EE Application Server. Dadurch können bewährte Legacy-Anwendungen in neue Geschäftsprozesse integriert werden.
- Mit WebTransactions steht in openSEAS ein Produkt zur Verfügung, welches es ermöglicht, bewährte Host-Anwendungen flexibel in neuen Geschäftsprozessen und modernen Einsatzszenarien zu nutzen. Bestehende UTM-Anwendungen können unverändert ins Web übernommen werden.

1.1 Kurzbeschreibung des Produkts openUTM-Client

Das Produkt openUTM-Client bietet Client-Server-Kommunikation mit UTM-Server-Anwendungen, die auf Windows-Systemen, Unix-Systemen und BS2000/OSD-Systemen ablaufen. openUTM-Client gibt es mit den Trägersystemen UPIC und OpenCPIC. Das Trägersystem hat die Aufgabe, die Verbindung zu anderen benötigten Systemkomponenten (z.B. Transportsystem) herzustellen und die Client/Server-Kommunikation zu steuern.

Zum Aufruf von Services einer UTM-Server-Anwendung bietet openUTM-Client die standardisierten X/Open-Schnittstellen CPI-C und XATMI. CPI-C und XATMI werden sowohl vom Trägersystem UPIC als auch vom Trägersystem OpenCPIC unterstützt.

- CPI-C steht für **C**ommon **P**rogramming **I**nterface for **C**ommunication.
  Mit CPI-C wurde eine Untermenge der Funktionen der in X/OPEN definierten Schnittstelle CPI-C realisiert. CPI-C ermöglicht Client-Server-Kommunikation zwischen einer CPI-C-Client-Anwendung und Services einer UTM-Anwendung, die entweder die CPI-C- oder die KDCS-Schnittstelle nutzen.

- XATMI ist eine X/OPEN-Schnittstelle für einen Communication Resource Manager, mit dem Client-Server-Kommunikation mit fernen UTM-Server-Anwendungen realisiert werden kann. XATMI ermöglicht die Kommunikation mit den Services einer UTM-Anwendung, die die XATMI-Server-Schnittstelle nutzen.
  XATMI ist die in der X/OPEN Preliminary Specification definierte Schnittstelle.

openUTM-Client für verschiedene Plattformen

openUTM-Client gibt es für folgende Plattformen:
- Windows-Plattformen
- Unix-Plattformen
- BS2000/OSD (nur Trägersystem UPIC)

Da die Schnittstellen CPI-C und XATMI standardisiert, d.h. auf allen Plattformen identisch sind, können die auf einer der Plattformen erstellten und getesteten Client-Anwendungen auf jede der anderen Plattformen portiert werden.

1.2 Zielgruppe und Konzept des Handbuchs

Dieses Handbuch richtet sich an Organisatoren, Einsatzplaner, Programmierer und Administratoren, die auf UPIC basierende Clients für die Kommunikation mit UTM-Server-Anwendungen erstellen und nutzen wollen. Dieses Handbuch beschreibt also openUTM-Client nur für das Trägersystem UPIC. Informationen zum Trägersystem OpenCPIC finden Sie in einem gesonderten Handbuch „openUTM-Client für Trägersystem OpenCPIC“.

Die Beschreibung in diesem Handbuch gilt für die Windows-Plattformen, die Unix- und Linux-Plattformen sowie BS2000/OSD.
Spezielle Informationen, die sich nur auf eine bestimmte Plattform beziehen, sind durch entsprechende Überschriften als solche gekennzeichnet.



Wenn im Folgenden allgemein von Unix-System bzw. Unix-Plattform die Rede ist, dann ist darunter sowohl ein Unix-basiertes Betriebssystem wie z.B. Solaris oder HP-UX als auch eine Linux-Distribution wie z.B. SUSE oder Red Hat zu verstehen.

Wenn im Folgenden von Windows-System bzw. Windows-Plattform die Rede ist, dann sind damit alle Windows-Varianten gemeint, auf denen openUTM zum Ablauf kommt.

1.3 Wegweiser durch die Dokumentation zu openUTM

In diesem Abschnitt erhalten Sie einen Überblick über die Handbücher zu openUTM und zum Produktumfeld von openUTM.

1.3.1 openUTM-Dokumentation

Die openUTM-Dokumentation besteht aus Handbüchern, einer Online-Hilfe für den grafischen Administrationsarbeitsplatz openUTM WinAdmin sowie einer Freigabemitteilung für jede Plattform, auf der openUTM freigegeben wird.

Es gibt Handbücher, die für alle Plattformen gültig sind, sowie Handbücher, die jeweils für BS2000/OSD bzw. für Unix-Systeme und Windows-Systeme gelten.

Sämtliche Handbücher sind als PDF-Datei im Internet verfügbar unter der Adresse

http://manuals.ts.fujitsu.com

Geben Sie dort in das Feld **Produktsuche** den Suchbegriff "openUTM V6.1" ein, um alle openUTM-Handbücher der Version 6.1 anzeigen zu lassen.

Die Handbücher sind auf offenen Plattformen auf der Enterprise DVD enthalten und stehen außerdem auch auf der WinAdmin DVD (für BS2000/OSD) zur Verfügung.

Die folgenden Abschnitte geben einen Aufgaben-bezogenen Überblick über die Dokumentation zu openUTM V6.1. Eine vollständige Liste der Dokumentation zu openUTM finden Sie im Literaturverzeichnis auf Seite 415.

Einführung und Überblick

Das Handbuch **Konzepte und Funktionen** gibt einen zusammenhängenden Überblick über die wesentlichen Funktionen, Leistungen und Einsatzmöglichkeiten von openUTM. Es enthält alle Informationen, die Sie zum Planen des UTM-Einsatzes und zum Design einer UTM-Anwendung benötigen. Sie erfahren, was openUTM ist, wie man mit openUTM arbeitet und wie openUTM in die BS2000/OSD-, Unix- und Windows-Plattformen eingebettet ist.

Programmieren

- Zum Erstellen von Server-Anwendungen über die KDCS-Schnittstelle benötigen Sie das Handbuch **Anwendungen programmieren mit KDCS für COBOL, C und C++**, in dem die KDCS-Schnittstelle in der für COBOL, C und C++ gültigen Form beschrieben ist. Diese Schnittstelle umfasst sowohl die Basisfunktionen des universellen Transaktionsmonitors als auch die Aufrufe für verteilte Verarbeitung. Es wird auch die Zusammenarbeit mit Datenbanken beschrieben.
- Wollen Sie die X/Open-Schnittstellen nutzen, benötigen Sie das Handbuch **Anwendungen erstellen mit X/Open-Schnittstellen**. Es enthält die UTM-spezifischen Ergänzungen zu den X/Open-Programmschnittstellen TX, CPI-C und XATMI sowie Hinweise zu Konfiguration und Betrieb von UTM-Anwendungen, die X/Open-Schnittstellen nutzen. Ergänzend dazu benötigen Sie die X/Open-CAE-Specification für die jeweilige X/Open-Schnittstelle.
- Wenn Sie Daten auf Basis von XML austauschen wollen, benötigen Sie das Dokument **XML für openUTM**. Darin werden die C- und COBOL-Aufrufe beschrieben, die zum Bearbeiten von XML-Dokumenten benötigt werden.
- Für BS2000/OSD gibt es Ergänzungsbände für die Programmiersprachen Assembler, Fortran, Pascal-XT und PL/1.

Konfigurieren

Zur Definition von Konfigurationen steht Ihnen das Handbuch **Anwendungen generieren** zur Verfügung. Darin ist beschrieben, wie Sie mit Hilfe des UTM-Tools KDCDEF sowohl für eine stand-alone UTM-Anwendung als auch für eine UTM-Cluster-Anwendung

- die Konfiguration definieren
- die KDCFILE erzeugen
- und im Falle einer UTM-Cluster-Anwendung die UTM-Cluster-Dateien erzeugen.

Zusätzlich wird gezeigt, wie Sie wichtige Verwaltungs- und Benutzerdaten mit Hilfe des Tools KDCUPD in eine neue KDCFILE übertragen, z.B. beim Umstieg auf eine neue Version von openUTM oder nach Änderungen in der Konfiguration. Für eine UTM-Cluster-Anwendung wird außerdem gezeigt, wie Sie diese Daten mit Hilfe des Tools KDCUPD in die neuen UTM-Cluster-Dateien übertragen.

Binden, Starten und Einsetzen

Um UTM-Anwendungen einsetzen zu können, benötigen Sie für das betreffende Betriebssystem (BS2000/OSD bzw. Unix-/Windows-Systeme) das Handbuch **Einsatz von openUTM-Anwendungen**.
Dort ist beschrieben, wie man ein UTM-Anwendungsprogramm bindet und startet, wie man sich bei einer UTM-Anwendung an- und abmeldet und wie man Anwendungsprogramme strukturiert und im laufenden Betrieb austauscht. Außerdem enthält es die UTM-Kommandos, die dem Terminal-Benutzer zur Verfügung stehen. Zudem wird ausführlich auf die Punkte eingegangen, die beim Betrieb von UTM-Cluster-Anwendungen zu beachten sind.

Administrieren und Konfiguration dynamisch ändern

- Für das Administrieren von Anwendungen finden Sie die Beschreibung der Programmschnittstelle zur Administration und die UTM-Administrationskommandos im Handbuch **Anwendungen administrieren**. Es informiert über die Erstellung eigener Administrationsprogramme für den Betrieb einer stand-alone UTM-Anwendung oder einer UTM-Cluster-Anwendung sowie über die Möglichkeiten, mehrere UTM-Anwendungen zentral zu administrieren. Darüber hinaus beschreibt es, wie Sie Message Queues und Drucker mit Hilfe der KDCS-Aufrufe DADM und PADM administrieren können.
- Wenn Sie den grafischen Administrationsarbeitsplatz **openUTM WinAdmin** einsetzen, dann steht Ihnen folgende Dokumentation zur Verfügung:
  - Die **WinAdmin-Beschreibung** bietet einen umfassenden Überblick über den Funktionsumfang von WinAdmin und das Handling von WinAdmin. Dieses Dokument wird mit der Software ausgeliefert und ist zusätzlich auch online als PDF-Datei verfügbar.
  - Das **Online-Hilfesystem** beschreibt kontextsensitiv alle Dialogfelder und die zugehörigen Parameter, die die grafische Oberfläche bietet. Außerdem wird dargestellt, wie man WinAdmin konfiguriert, um stand-alone UTM-Anwendungen und UTM-Cluster-Anwendungen administrieren zu können.

Testen und Fehler diagnostizieren

Für die o.g. Aufgaben benötigen Sie außerdem die Handbücher **Meldungen, Test und Diagnose** (jeweils ein Handbuch für Unix-/Windows-Systeme und für BS2000/OSD). Sie beschreiben das Testen einer UTM-Anwendung, den Inhalt und die Auswertung eines UTM-Dumps, das Verhalten im Fehlerfall, das Meldungswesen von openUTM, sowie alle von openUTM ausgegebenen Meldungen und Returncodes.

openUTM-Clients erstellen

Wenn Sie Client-Anwendungen für die Kommunikation mit UTM-Anwendungen erstellen wollen, stehen Ihnen folgende Handbücher zur Verfügung:

- Das Handbuch **openUTM-Client für Trägersystem UPIC** beschreibt Erstellung und Einsatz von Client-Anwendungen, die auf UPIC basieren. Neben der Beschreibung der Schnittstellen CPI-C und XATMI erhalten Sie Informationen, wie Sie die C++-Klassen für die schnelle und einfache Programmerstellung nutzen können.
- Das Handbuch **openUTM-Client für Trägersystem OpenCPIC** beschreibt, wie man OpenCPIC installiert und konfiguriert. Es zeigt auf, was beim Programmieren einer CPI-C-Anwendung zu beachten ist und welche Einschränkungen es gegenüber der Programmschnittstelle X/Open CPI-C gibt.
- Für die mit **BeanConnect** ausgelieferten **JUpic-Java-Klassen** wird die Dokumentation mit der Software ausgeliefert. Diese Dokumentation besteht aus Word- und PDF-Dateien, die die Einführung und die Installation beschreiben, sowie aus einer Java-Dokumentation mit der Beschreibung der Java-Klassen.
- Das Handbuch **BizXML2Cobol** beschreibt, wie Sie bestehende Cobol-Programme einer UTM-Anwendung so erweitern können, dass sie als Standard-Web-Service auf XML-Basis genutzt werden können. Die Arbeit mit der grafischen Bedienoberfläche ist in der zugehörigen **Online-Hilfe** beschrieben.
- Wenn Sie UTM-Services auf einfache Weise ins Web stellen möchten, benötigen Sie das Handbuch **Web-Services für openUTM**. Das Handbuch beschreibt, wie Sie mit dem Software-Produkt WS4UTM (WebServices for openUTM) Services von UTM-Anwendungen als Web Services verfügbar machen. Die Arbeit mit der grafischen Bedienoberfläche ist in der zugehörigen **Online-Hilfe** beschrieben.

Kopplung mit der IBM-Welt

Wenn Sie aus Ihrer UTM-Anwendung mit Transaktionssystemen von IBM kommunizieren wollen, benötigen Sie außerdem das Handbuch **Verteilte Transaktionsverarbeitung zwischen openUTM und CICS-, IMS- und LU6.2-Anwendungen**. Es beschreibt die CICS-Kommandos, IMS-Makros und UTM-Aufrufe, die für die Kopplung von UTM-Anwendungen mit CICS- und IMS-Anwendungen benötigt werden. Die Kopplungsmöglichkeiten werden anhand ausführlicher Konfigurations- und Generierungsbeispiele erläutert. Außerdem beschreibt es die Kommunikation über openUTM-LU62, sowie dessen Installation, Generierung und Administration.

1.3.2 Dokumentation zum openSEAS-Produktumfeld

Die Verbindung von openUTM zum openSEAS-Produktumfeld wird im openUTM-Handbuch **Konzepte und Funktionen** kurz dargestellt. Die folgenden Abschnitte zeigen, welche der openSEAS-Dokumentationen für openUTM von Bedeutung sind.

Integration von J2EE Application Servern und UTM-Anwendungen

Der Adapter BeanConnect gehört zur Produkt-Suite openSEAS. Der BeanConnect-Adapter realisiert die Verknüpfung zwischen klassischen Transaktionsmonitoren und J2EE Application Servern und ermöglicht damit die effiziente Integration von Legacy-Anwendungen in Java-Anwendungen.

- Das Handbuch **BeanConnect** beschreibt das Produkt BeanConnect, das einen JCA 1.5-konformen Adapter bietet, der UTM-Anwendungen mit Anwendungen auf Basis von J2EE, z.B. mit dem Application Server von Oracle, verbindet.

  Die Handbücher zum Application Server von Oracle sind bei Oracle beziehbar.

Web-Anbindung und Anwendungsintegration

Zum Anschließen neuer und bestehender UTM-Anwendungen an das Web mit dem Produkt WebTransactions benötigen Sie die Handbücher **WebTransactions**.

Die Dokumentation wird durch JavaDocs ergänzt.

1.3.3 Readme-Dateien

Funktionelle Änderungen und Nachträge der aktuellen Produktversion zu diesem Handbuch entnehmen Sie bitte ggf. den Produkt-spezifischen Readme-Dateien.

Readme-Datei online

Readme-Dateien stehen Ihnen online bei dem jeweiligen Produkt zusätzlich zu den Produkthandbüchern unter *http://manuals.ts.fujitsu.com* zur Verfügung.

Readme-Datei unter BS2000/OSD

Auf Ihrem BS2000-System finden die Readme-Dateien für die installierten Produkte unter dem Dateinamen:

```
SYSRME.<produkt>.<version>.D
```

Die Benutzerkennung, unter der sich die Readme-Datei befindet, erfragen Sie bitte bei Ihrer zuständigen Systembetreuung. Den vollständigen Pfadnamen erhalten Sie auch mit folgendem Kommando:

```
/SHOW-INSTALLATION-PATH INSTALLATION-UNIT=<produkt>,LOGICAL-ID=SYSRME.D
```

Sie können die Readme-Datei am Bildschirm mit dem Kommando `/SHOW-FILE` oder mit einem Editor ansehen oder auf einem Standarddrucker mit folgendem Kommando ausdrukken:

```
/PRINT-DOCUMENT <dateiname>, LINE-SPACING=*BY-EBCDIC-CONTROL
```

Readme-Datei unter Unix-Systemen:

Die Readme-Datei und ggf. weitere Dateien wie z.B. eine Handbuchergänzungsdatei finden Sie im *utmpfad* unter `/docs/`*sprache*.

Readme-Datei unter Windows-Systemen:

Die Readme-Datei und ggf. weitere Dateien wie z.B. eine Handbuchergänzungsdatei finden Sie im *utmpfad* unter `\Docs\`*sprache*.

1.4 Änderungen gegenüber dem Vorgängerhandbuch

Das Das Handbuch openUTM-Client V6.1 für Trägersystem UPIC enthält gegenüber dem Handbuch openUTM-Client V6.0 für Trägersystem UPIC folgende Neuerungen.

Einige der Änderungen sind für alle Clients relevant, d.h. sie gelten sowohl für Clients von stand-alone UTM-Anwendungen als auch für Clients von UTM-Cluster-Anwendungen. Andere Änderungen sind nur für Clients von UTM-Cluster-Anwendungen relevant.

Neue und geänderte Funktionen für alle UPIC-Clients

- Shutdown WARN oder GRACE in einer UTM-Anwendung

  Eine UPIC-Client-Anwendung kann gezielt auf ein Shutdown WARN oder GRACE reagieren. Falls die Verbindung zu einer UTM-Anwendung aufgebaut ist und in der Anwendung danach ein Shutdown WARN oder GRACE eingeleitet wurde, dann wird dies über das UPIC-Protokoll an den Client weitergegeben. Das Client-Programm kann den Shutdown-Status und die Shutdown-Zeit über die neuen Funktionen *Extract_Shutdown_State* und *Extract_Shutdown_Time* abfragen.

- Wegfall der OCX-Schnittstelle

  Die OCX-Schnittstelle wird in UPIC nicht mehr angeboten. Damit entfällt das bisherige Kapitel "ActiveX Control UpicB.ocx".

- Plattform Linux-x86 64-Bit

  Der Ablauf von UPIC-Client-Anwendungen auf Linux-x86 Systemen ist sowohl in einer 32-Bit als auch in einer 64-Bit Umgebung möglich.

Neue Funktionen für UPIC-Clients von UTM-Cluster-Anwendungen

Voraussetzung für die Funktionalität ist, dass Cluster-Einträge (Präfix CD) der Upicfile verwendet werden.

- Shutdown WARN oder GRACE in einer UTM-Cluster-Anwendung
  - Wenn der UPIC-Client nach dem Verbindungsaufbau zu einer Knoten-Anwendung feststellt, dass die Knoten-Anwendung sich bereits im Zustand Shutdown WARN oder GRACE befindet, dann baut er die Verbindung wieder ab und baut sie zu einer anderen Knoten-Anwendung auf.
  - Ist ein UPIC-Client bereits an einer Knoten-Anwendung angemeldet, dann kann er den Shutdown-Status und ggf. den Shutdown-Zeitpunkt anfragen (neue CPI-C-Aufrufe *Extract_Shutdown_State* und *Extract_Shutdown_Time*).

- Knoten-Anwendung bei offenem Vorgang wechseln

  Falls die Anmeldung eines Benutzers an einer Knoten-Anwendung abgelehnt wird, weil für diesen Benutzer ein offener Vorgang existiert, der an eine andere Knoten-Anwendung gebunden ist, dann versucht UPIC, den Benutzer an dieser (anderen) Knoten-Anwendung anzumelden.

1.5 Darstellungsmittel

Symbole

Beschreibungsteile, die nur für bestimmte Plattformen von UPIC gelten, sind wie folgt durch ein Symbol am linken Rand gekennzeichnet:

B
B
BS2000/OSD-spezifische Teile der Beschreibung sind am linken Rand mit dem nebenstehenden Symbol gekennzeichnet.

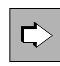
X
X
Unix-System-spezifische Teile der Beschreibung sind am linken Rand mit dem nebenstehenden Symbol gekennzeichnet.

W
W
Windows-System-spezifische Teile der Beschreibung sind am linken Rand mit dem nebenstehenden Symbol gekennzeichnet.

B/X
B/X
Teile der Beschreibung, die nur für openUTM in BS2000/OSD und Unix-Systemen von Bedeutung sind, sind am linken Rand mit dem nebenstehenden Symbol gekennzeichnet.

B/W
B/W
B/W
Teile der Beschreibung, die nur für openUTM in BS2000/OSD und in Windows-Systemen von Bedeutung sind, sind am linken Rand mit dem nebenstehenden Symbol gekennzeichnet.

B/X
B/X
Teile der Beschreibung, die nur für openUTM in Unix- und Windows-Systemen von Bedeutung sind, sind am linken Rand mit dem nebenstehenden Symbol gekennzeichnet.


für Verweise auf umfassende und detaillierte Informationen zum jeweiligen Thema.


für Hinweistexte.


für Warnhinweise.

Metasyntax

Die in diesem Handbuch verwendete Metasyntax können Sie der folgenden Tabelle entnehmen:

Formale Darstellung	Erläuterung	Beispiel
GROSSBUCHSTABEN	Großbuchstaben bezeichnen Konstanten (Namen von Aufrufen, Anweisungen, Feldnamen, Kommandos und Operanden etc.), die in dieser Form anzugeben sind.	`LOAD-MODE=STARTUP`
kleinbuchstaben	In Kleinbuchstaben sind in Syntaxdiagrammen und Operandenbeschreibung die Platzhalter für Operandenwerte dargestellt.	`KDCFILE=filebase`
kleinbuchstaben	Im Fließtext werden Variablen, Namen von Datenstrukturen und Feldern, Schlüsselwörter (z.B. C-Befehle, Unix- und Windows-Dateinamen etc.) in kursiven Kleinbuchstaben dargestellt.	Für COBOL sind die Datenstrukturen im COPY-Element KCINIC definiert, Datenstrukturen für C/C++ in der Include-Datei *kcini.h*.
{ } und |	In geschweiften Klammern stehen alternative Angaben, von denen Sie eine auswählen müssen. Die zur Verfügung stehenden Alternativen werden jeweils durch einen Strich getrennt aufgelistet.	`STATUS={ ON | OFF }`
[ ]	In eckigen Klammern stehen wahlfreie Angaben, die entfallen können.	`KDCFILE=( filebase [, { SINGLE| DOUBLE} ] )`
( )	Kann für einen Operanden eine Liste von Parametern angegeben werden, sind diese in runde Klammern einzuschließen und durch Kommata zu trennen. Wird nur ein Parameter angegeben, kann auf die Klammern verzichtet werden.	KEYS=(key1,key2,...key*n*)
<u>Unterstreichen</u>	Unterstreichen kennzeichnet den Standardwert.	CONNECT= { A/YES | <u>NO</u> }

Tabelle 1: Metasyntax

Formale Darstellung	Erläuterung	Beispiel
Kurzform	Die Standardkurzform für Anweisungen, Operanden und Operandenwerte wird „fett" hervorgehoben. Die Kurzform kann alternativ angegeben werden.	`TRANSPORT-SELECTOR=c'C'`

. . .	Punkte zeigen die Wiederholbarkeit einer syntaktischen Einheit an. Außerdem kennzeichnen die Punkte Ausschnitte aus einem Programm, einer Syntaxbeschreibung o.ä.	`KDCDEF starten`
		`:`
		`:`
		`OPTION DATA=statement_file`
		`:`
		`END`

Tabelle 1: Metasyntax

2 Anwendungsbereich

Da die Oberflächengestaltung keine eigentliche Aufgabe eines Transaktionsmonitors ist, wird sie aus der UTM-Anwendung in Clients ausgelagert. Die UTM-Anwendung stellt damit den Server dar. openUTM-Client mit den Schnittstellen CPI-C und XATMI bietet Ihnen die Möglichkeit, Client-Programme zu erstellen, die mit der UTM-Anwendung als Server zusammenarbeiten.

Sie können Client-Programme aber auch als Auftragstreiber verwenden.

Das Client-Server-Konzept

Das Client-Server-Konzept hat zum Ziel, den einzelnen Anwendern in einem Netz Dienste (=Services, z.B. Daten, Programme, Geräte) verfügbar zu machen und die Stärken der einzelnen Systeme optimal zu nutzen.

Das Client-Server-Konzept wird immer dann verwendet, wenn viele Anforderer (Clients) vorhanden sind, die dieselbe Dienstleistung (Service) benötigen. Eine Analogie zum Client-Server-Konzept ist folgende: Der Prozedur- oder Unterprogramm-Aufruf stellt eine Client-Server-Beziehung zwischen Haupt- und Unterprogramm her. Mit dem einen Unterschied, dass die aufgerufene Prozedur jetzt entfernt vom „Client“ arbeitet.

Clients (Nutzer von Diensten) können Leistungen und Informationen von allen Servern im Netz anfordern.

Für Server (Anbieter von Diensten) gilt: Es werden Leistungen angeboten, wobei die gemeinsam genutzten Informationsquellen wie Dateien und Datenbanken innerhalb einer Netzkonfiguration beliebig verteilt sein können.

2.1 Das Konzept von openUTM-Client

Zum Aufruf von Services bietet openUTM-Client standardisierte X/Open-Schnittstellen auf unterschiedlichen Plattformen und Trägersystemen.

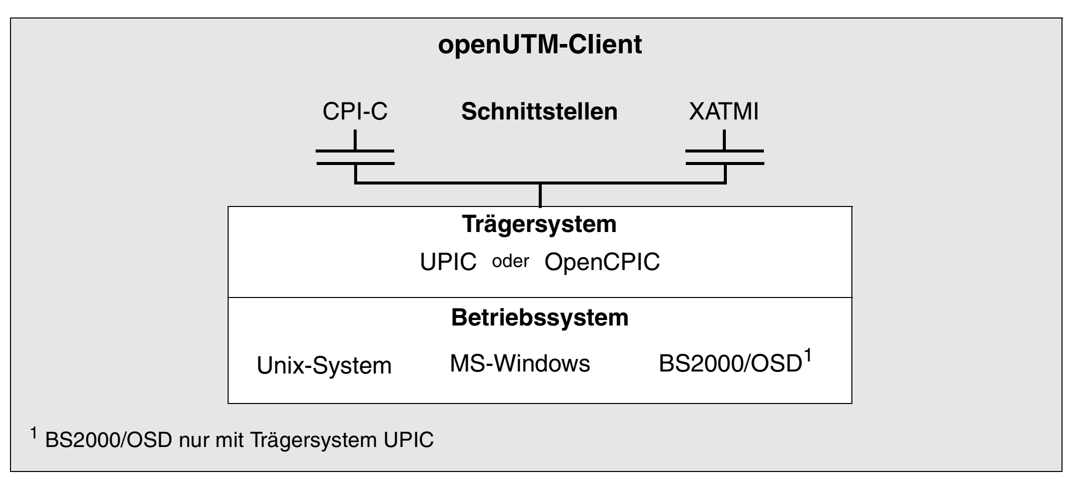


Bild 1: Standardisierte X/Open-Schnittstellen

Schnittstellen

openUTM-Client kann mit den X/Open-Schnittstellen CPI-C und XATMI programmiert werden.

X/W *Einschalung der CPI-C-Schnittstelle für Windows- und Unix-Systeme mit Trägersystem UPIC:*

X/W Für Windows- und Unix-Systeme stellt openUTM-Client (Trägersystem UPIC) eine
X/W Einschalung der CPI-C-Schnittstelle zur Verfügung: die Wrapper Class CUpic.

X/W Für jedes CUpic Objekt wird ein Worker Thread erzeugt. So können in einem Anwendungs-
X/W programm mehrere UPIC-Conversations parallel aktiv sein. Die Erzeugung und Steuerung
X/W der Threads wird von der Klasse CUpic transparent erledigt.

X/W Nähere Informationen zur Klasse CUpic finden Sie in Kapitel „C++ Klasse CUpic“ auf
X/W Seite 33.

Trägersysteme

Die Schnittstellen CPI-C und XATMI werden sowohl vom Trägersystem UPIC als auch vom Trägersystem OpenCPIC zur Verfügung gestellt. Das Trägersystem hat die Aufgabe, die Verbindung zu den anderen benötigten Komponenten herzustellen wie z.B. dem Transportzugriffssystem (TCP/IP in Windows-, Unix-Systemen oder BS2000/OSD, PCMX-32 in Windows-Systemen, CMX/PCMX in Unix-Systemen oder BCAM in BS2000/OSD).

Das Trägersystem UPIC bietet gegenüber OpenCPIC folgende Vorteile:

- Das Client-Programm kann das Betätigen von Funktionstasten simulieren.
- Zwischen Client und Server können zusammen mit den Daten auch Formatkennzeichen als Strukturierungsinformationen ausgetauscht werden.
- Das Client Programm kann ein neues Passwort vergeben.

Betriebssystem-Plattformen

Ein Trägersystem kann auf den verschiedensten Plattformen residieren. Dies sind:

W
- Windows-Systeme

X
- alle gängigen Unix-Systeme

B
- BS2000/OSD-Systeme (nur Trägersystem UPIC)

Da die Schnittstellen CPI-C und XATMI standardisiert, d.h. auf allen Plattformen identisch sind, können die auf einer der Plattformen erstellten und getesteten Client-Anwendungen auf jede der anderen Plattformen portiert werden.

Begriffsfestlegung

Im Folgenden wird ein Programm, das CPI-C-Aufrufe enthält, als **CPI-C-Programm** und ein Programm, das XATMI-Aufrufe enthält, als **XATMI-Programm** bezeichnet. Das darunterliegende Trägersystem wird nur dann erwähnt, wenn es die Funktionalität beeinflusst oder an der Schnittstelle sichtbar ist.

Eine **CPI-C-Anwendung** bzw. **XATMI-Anwendung** ist die Gesamtheit von CPI-C- bzw. XATMI-Programmen und allen für das jeweilige Trägersystem notwendigen Konfigurationsdateien.

2.2 Client-Server-Kommunikation mit openUTM

Das folgende Bild veranschaulicht, über welche Schnittstellen openUTM-Clients mit einem UTM-Server kommunizieren können.

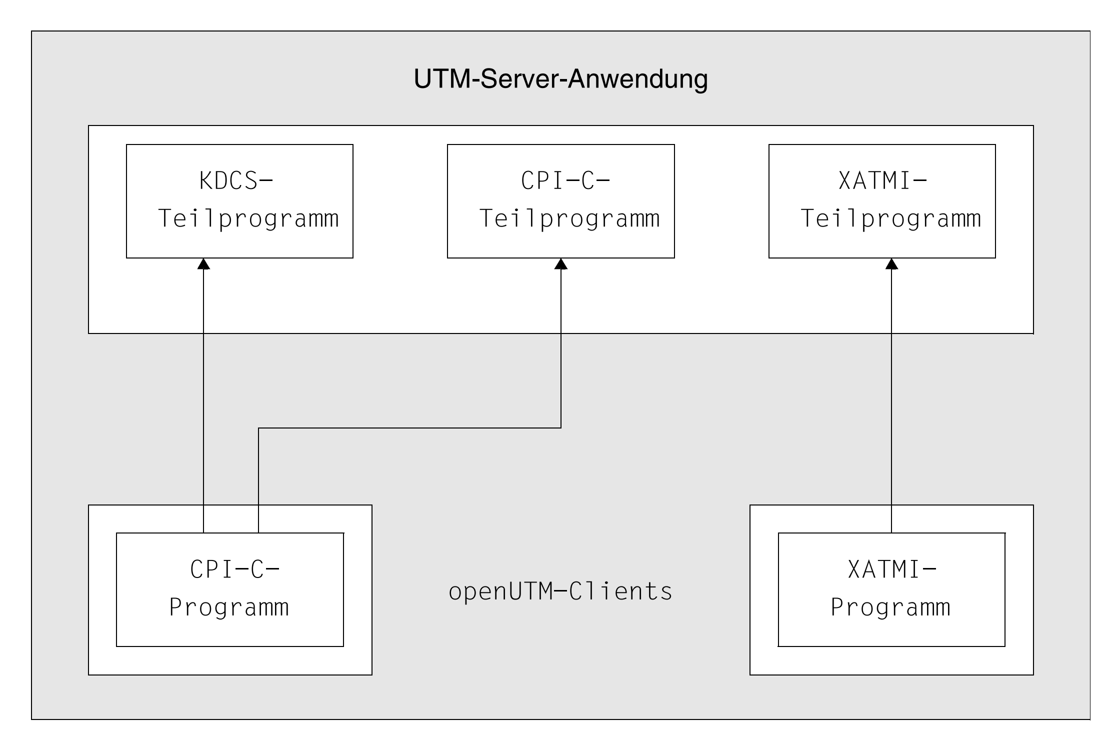


Bild 2: Schnittstellen zwischen UTM-Server und UTM-Clients

Ein Client mit CPI-C-Programm kann sowohl mit einem KDCS-Teilprogramm als auch mit einem CPI-C-Teilprogramm kommunizieren, ein Client mit XATMI-Programm kann immer nur ein XATMI-Teilprogramm als Service nutzen. Ein KDCS-Teilprogramm ist ein Teilprogramm eines UTM-Servers, das KDCS-Aufrufe enthält.

Client und Server können auf allen Plattformen auf dem gleichen Rechner liegen.

Im Folgenden wird eine UTM-Server-Anwendung immer mit UTM-Anwendung oder kurz mit openUTM bezeichnet.

2.3 UPIC-Local, UPIC-Remote und Multithreading

Mit UPIC als Trägersystem haben Sie zwei prinzipielle Möglichkeiten, Client-Programme zu koppeln: UPIC-Local (Unix-/Windows-Systeme) und UPIC-Remote (alle Plattformen).

Die Informationen in diesem Handbuch gelten, wenn nicht anders vermerkt, für beide Varianten.

UPIC-Local (Unix- / Windows-Systeme)

Mit UPIC-Local (UPIC-L) können Sie ein Client-Programm lokal mit einer UTM-Anwendung auf dem gleichen Unix- oder Windows-System koppeln. Das Trägersystem UPIC-Local gibt es für Unix- und Windows-Systeme. Es ist in die UTM-Server-Software integriert. Für die Kopplung über UPIC-Local benötigen Sie also weder das Produkt openUTM-Client noch die Kommunikationskomponente PCMX.

Diese Möglichkeit gibt es nur auf einem Unix- und Windows-System.

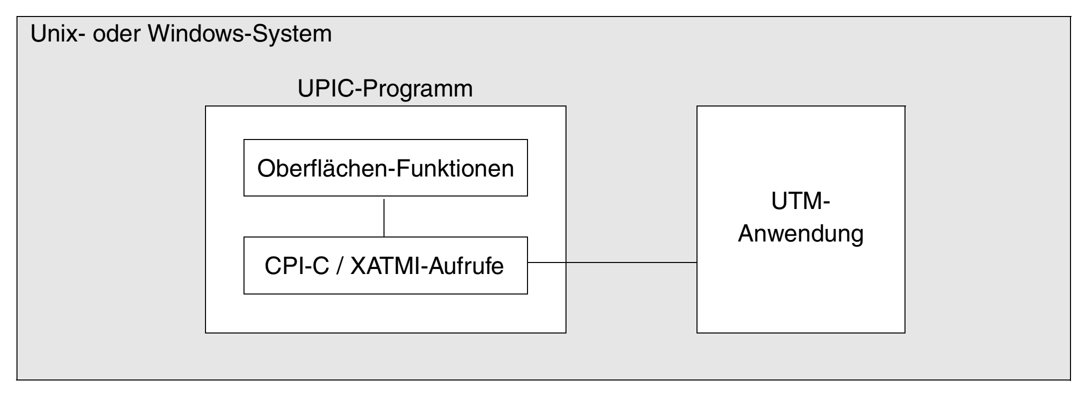


Bild 3: Lokaler Anschluss an eine UTM-Anwendung

Die Oberflächen-Funktionen stellen eine benutzerfreundliche Oberfläche zur Verfügung. Über die CPI-C- oder XATMI-Aufrufe kommuniziert das Client-Programm mit der UTM-Anwendung. Dabei werden nur Nettodaten übermittelt.

UPIC-Remote

Mit UPIC-Remote (UPIC-R) können Sie ein Client-Programm mit UTM-Anwendungen koppeln, die auf einem beliebigen Rechner im Netz laufen. Diese Möglichkeit gibt es für alle Server-Plattformen (Windows-, Unix-Systeme und BS2000/OSD). Sie benötigen hierfür das Produkt openUTM-Client. openUTM-Client enthält UPIC-Remote in zwei verschiedenen Ausführungen. In der einen Variante wird TCP/IP über die Socket-Schnittstelle verwendet. Zusätzliche Kommunikationskomponenten sind hierfür nicht notwendig. Bei der klassischen Variante wird der Zugriff aufs Netz über die Plattform-spezifischen Kommunikationskomponenten PCMX oder CMX geregelt (siehe Bild 4).

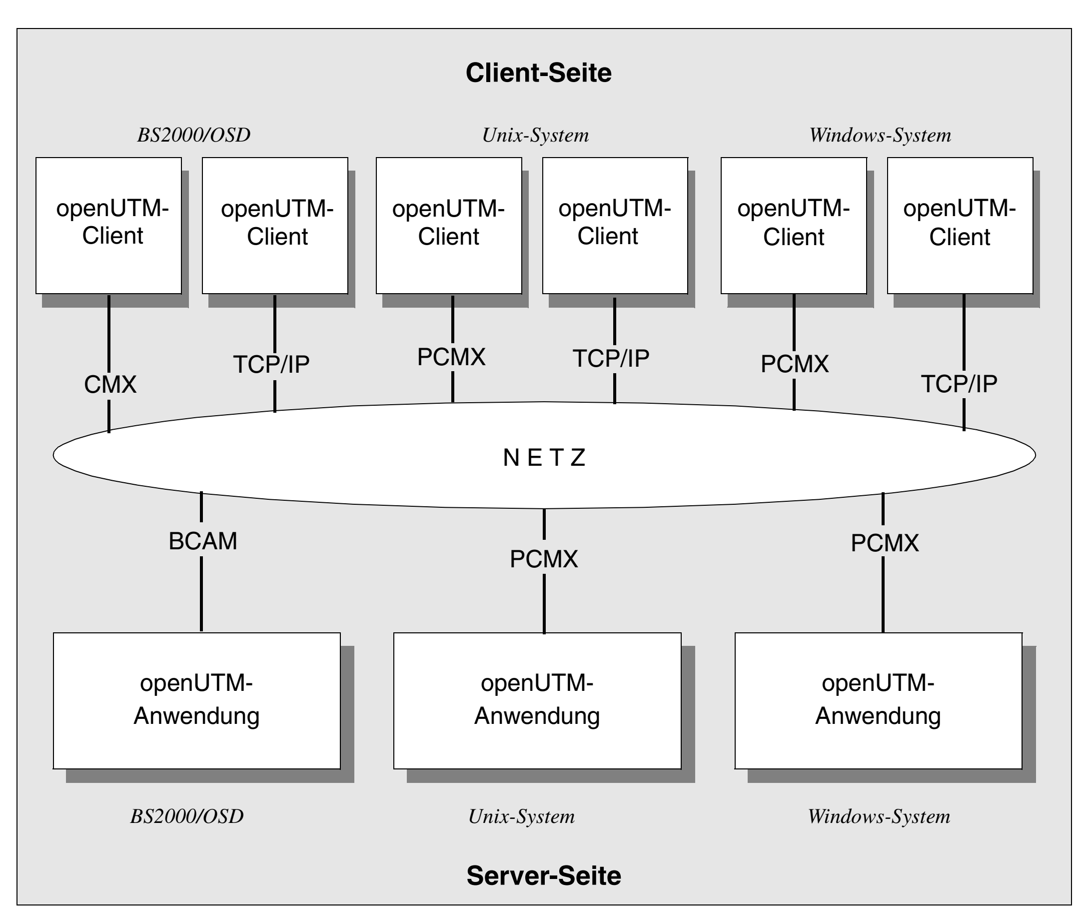


Bild 4: Remote-Anschluss an UTM-Anwendungen

Auch beim Remote-Anschluss ist es möglich, dass das Client-Programm und die UTM-Anwendung auf dem gleichen Rechner liegen. Die Kommunikation zwischen Client-Programm und openUTM wird aber auch in diesem Fall über die Kommunikationskomponenten TCP/IP, PCMX oder CMX abgewickelt.

Multithreading

Das Trägersystem UPIC ist grundsätzlich multithreadingfähig. Ob Sie diese Fähigkeit in Ihrer Anwendung nutzen können, ist von zwei Komponenten abhängig:

- das Betriebssystem muss Multithreading unterstützen
- das verwendete Kommunikationssystem muss Multithreading unterstützen

So sieht es bei UPIC auf den einzelnen Plattformen aus:

X/W
- UPIC-L ist nicht multithreadingfähig

W
- UPIC-R auf Windows-Systemen ist uneingeschränkt multithreadingfähig

X
- UPIC-R auf Unix-Systemen ist multithreadingfähig

B
- UPIC-R auf BS2000/OSD ist nicht multithreadingfähig

Die genauen Angaben entnehmen Sie bitte der jeweiligen Freigabemitteilung.

2.4 Unterstützung von UTM-Cluster-Anwendungen

Ein openUTM-Client mit UPIC als Trägersystem kann mit einer UTM-Cluster-Anwendung ebenso kommunizieren wie mit einer stand-alone UTM-Anwendung.

Ein Cluster ist eine Anzahl von Rechnern (Knoten), die über ein schnelles Netzwerk verbunden sind. Auf einem Cluster läuft openUTM in Form einer UTM-Cluster-Anwendung. Physikalisch gesehen besteht eine UTM-Cluster-Anwendung aus mehreren identisch generierten UTM-Anwendungen, den Knoten-Anwendungen, die auf den einzelnen Knoten laufen.

Der Client benötigt eine Liste der zugehörigen Knoten-Anwendungen. Aus dieser Liste wird dann zufällig eine Knoten-Anwendung ausgewählt, mit der die nächste Kommunikation erfolgen soll.

Wenn die Kommunikation mit dieser ausgewählten Knoten-Anwendung nicht möglich ist, wird automatisch ein Verbindungsaufbau mit der nächsten Knoten-Anwendung aus der Liste versucht. Dieser Vorgang wird so lange wiederholt, bis eine Kommunikation zu einer laufenden Knoten-Anwendung aufgebaut werden kann bzw. bis erkannt wird, dass alle Knoten-Anwendungen aus der Liste nicht erreichbar sind.

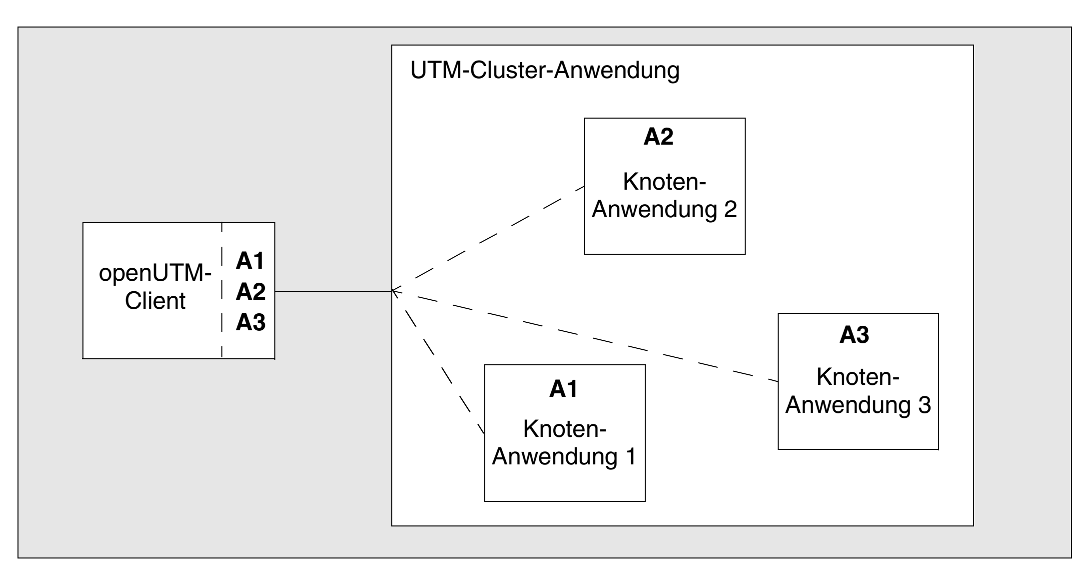


Bild 5: Kommunikation mit einer UTM-Cluster-Anwendung

Die Liste der Knoten-Anwendungen für jede UTM-Cluster-Anwendung wird in der Side Information-Datei (`upicfile`) übergeben. Details siehe Abschnitt „Side Information für UTM-Cluster-Anwendungen“ auf Seite 305.

3 C++ Klasse CUpic

In diesem Kapitel finden Sie Informationen über:

– Helper Classes
– Class CUpic

Zunächst wird jedoch eine kleine Einführung über die Konfiguration von UPIC gegeben.

3.1 Einleitung

Die CUpic Klasse ist eine Wrapper Class für das openUTM-Client Interface. Um diese Klasse verwenden zu können, muss UPIC korrekt konfiguriert werden.

Es gibt zwei Möglichkeiten der Konfiguration:

– Konfiguration mittels Helper Classes `CUpicLocAddr` und `CUpicRemAddr`
– Konfiguration von außen, mit einer Side Information-Datei (`upicfile`)

3.1.1 Konfiguration mittels Helper Classes CUpicLocAddr und CUpicRemAddr

Im einfachsten Fall werden die Konstruktoren benutzt, z.B.

```
CUpicLocAddr("upicw32")
```

und

```
CUpicRemAddr("        ", "sample", "local", 30000)
```

3.1.2 Konfiguration mit einer Side Information Datei (upicfile)

Bei der Konfiguration von außen muss eine `upicfile` vorhanden sein, die zumindest die Default-Einträge für einen lokalen Namen und für den *Symbolic_Destination_Name* enthält, z.B.:

```
LN.DEFAULT upicw32;
```

```
SD.DEFAULT sample.local hello PORT=30000;
```

Eine Beschreibung zur Konfiguration der `upicfile` finden Sie in Abschnitt „Die Side Information-Datei (upicfile)“ auf Seite 297.

3.1.3 Die CUpic Klasse auf threadfähigen Systemen

Auf threadfähigen Systemen (siehe dazu „Multithreading“ auf Seite 31) wird für jedes CUpic-Objekt ein Worker Thread erzeugt. So können in einem Anwendungsprogramm mehrere UPIC-Conversations parallel aktiv sein. Die Erzeugung und Steuerung der Threads wird von der Klasse CUpic transparent erledigt.

3.2 Helper Classes

Die Helper Classes definieren Adress- und Security-Objekte. Die Adress-Objekte können als Argumente der Konstruktoren der CUpic-Objekte übergeben werden. Security-Objekte können nur über die Property Handler Funktion

`SetSecurity()`

gesetzt werden.

3.2.1 CUpicLocAddr

`CUpicLocAddr` definiert eine lokale UPIC-Adresse.

3.2.1.1 Konstruktoren

CUpicLocAddr()

Der DEFAULT-Name des lokalen Namen in der Side Information wird verwendet.

CUpicLocAddr(const char * local_name)

`local_name`	wird dem *Enable_UTM_UPIC*-Aufruf als Argument übergeben.

CUpicLocAddr(const char * local_name
, const char * tsel_name)
, CM_INT32 port)

Die lokale RFC1006 Adresse wird explizit definiert.

`local_name`	wird dem *Enable_UTM_UPIC*-Aufruf als Argument übergeben. Der Wert NULL bedeutet, dass ein leerer *local_name* verwendet wird.
`tsel_name`	wird als direkter lokaler Name benutzt und dem Aufruf *Specify_Local_Tsel()* übergeben. Wenn der *tsel_name* nur Großbuchstaben und Ziffern enthält, dann wird das *Tsel_Format* TRANSDATA verwendet, ansonsten EBCDIC.
`port`	wird dem *Enable_UTM_UPIC*-Aufruf als Argument übergeben.

3.2.1.2 Member Functions

void SetTselName (const char * name)

`tsel_name` wird als direkter lokaler Name benutzt und dem Aufruf *Specify_Local_Tsel()* übergeben.
Wenn der *tsel_name* nur Großbuchstaben und Ziffern enthält, dann wird das *Tsel_Format* TRANSDATA verwendet, ansonsten EBCDIC.

void SetPort (CM_INT32 port)

`port` wird als lokale Portnummer benutzt und dem Aufruf *Specify_Local_Port()* übergeben.

int SetTselFormat (const unsigned char format)

Mit dieser Funktion kann das *Tsel_Format* gesetzt werden:

- ’A’ für ASCII
- ’E’ für EBCDIC
- ’T’ für TRANSDATA

3.2.2 CUpicRemAddr

`CUpicRemAddr` definiert eine entfernte UPIC-Adresse.

3.2.2.1 Konstruktoren

CUpicRemAddr ()

Der DEFAULT-Name für die entfernte Adresse wird benutzt.

CUpicRemAddr (const char * sym_dest_name)

`sym_dest_name`	wird dem Aufruf *Initialize_Conversation* übergeben.

CUpicRemAddr(const char * sym_dest_name
, const char * tsel_name
, const char * host_name
, CM_INT32 port
)

Die entfernte RFC1006 Adresse wird explizit definiert.

`sym_dest_name`	wird dem Aufruf *Initialize_Conversation* übergeben.
`tsel_name`	wird direkt als entfernter Name benutzt und dem Aufruf *Set_Partner_Tsel()* übergeben. Wenn der *tsel_name* nur Großbuchstaben und Ziffern enthält, dann wird das *Tsel_Format* TRANSDATA verwendet, ansonsten EBCDIC.
`host_name`	wird unmittelbar als entfernte Host-Adresse benutzt. Abhängig von dem benutzten Format des Strings wird er dem Aufruf *Set_Partner_Host_Name()* oder *Set_Partner_IP_Address()* übergeben.
`port`	wird als entfernte Portnummer benutzt und dem Aufruf *Set_Partner_Port()* übergeben.

3.2.2.2 Member Functions

void SetTselName (const char * tsel_name)

`tsel_name` wird direkt als entfernter Name benutzt und dem Aufruf *Set_Partner_Tsel()* übergeben.
Wenn der *tsel_name* nur Großbuchstaben und Ziffern enthält, dann wird das *Tsel_Format* TRANSDATA verwendet, ansonsten EBCDIC.

void SetHost (const char * host)

`host` wird unmittelbar als entfernte Host-Adresse benutzt. Abhängig von dem benutzten Format des Strings wird er dem Aufruf *Set_Partner_Host_Name()* oder *Set_Partner_IP_Address()* übergeben.

void SetPort (CM_INT32 port)

`port` wird als entfernte Portnummer benutzt und dem Aufruf *Set_Partner_Port()* übergeben.

int SetTselFormat (const unsigned char format)

Mit dieser Funktion kann das *Tsel_Format* gesetzt werden:

- ’A’ für ASCII
- ’E’ für EBCDIC
- ’T’ für TRANSDATA

3.2.3 CUpic Security

`CUpic Security` definiert die Security Attribute für UPIC.

CUpic Security ()

Es wird keine Security benutzt.

CUpic Security (char * uid)

`uid` wird dem Aufruf *Set_Conversation_Security_User_ID()* übergeben.

CUpic Security (char * uid, char * pwd)

`uid` wird dem Aufruf *Set_Conversation_Security_User_ID()* übergeben.

`pwd` wird dem Aufruf *Set_Conversation_Password()* übergeben.

3.3 ClassCUpic

Ein CUpic-Objekt repräsentiert eine Conversation mit einem UTM-Service.

3.3.1 Konstruktoren

CUpic()

Der DEFAULT-Name der lokalen und entfernten Adresse wird verwendet.

CUpic (CUpicLocAddr 1)

Die angegebene lokale Adresse und der DEFAULT-Name der entfernten Adresse werden verwendet.

CUpic (CUpicRemAddr 2)

Die angegebene entfernte Adresse und der DEFAULT-Name der lokalen Adresse werden verwendet.

CUpic (CUpicLocAddr 1, CUpicRemAddr 2)

Die angegebenen lokalen und entfernten Adressen werden verwendet.

3.3.2 Property Handlers

void SetLocal(CUpicLocAddr l)

Definiert eine neue lokale Adresse.

void SetRemote(CUpicRemAddr r)

Definiert eine neue entfernte Adresse.

void SetSecurity(CUpicSecurity s)

Definiert neue Security Attribute.

void SetEncryption(BOOL)

Aktiviert die Verschlüsselung.

void SetFunctionKey(CM_FUNCTION_KEY)

Schaltet den function key beim nächsten Senden ein.

void SetTPName(const char * name)

Setzt den Transaktionscode (TAC) für eine neue Conversation.

Die Funktion erwartet einen String mit der Länge strlen(name).

void SetMapName(const char * name)

Setzt den map name für folgende Sendeaufrufe.

Die Funktion erwartet einen String mit der Länge strlen(name).

void GetTPName(char * name)

Liest den momentan gültigen Transaktionscode (TAC).

Die Funktion kopiert einen String mit folgendem ’\0’ an die angegebene Adresse. Der Zielstring muss mindestens als `char name [9]` deklariert sein.

void GetMapName(char * name)

Liest den letzten erhaltenen map name aus.

Die Funktion kopiert einen String mit folgendem ’\0’ an die angegebene Adresse. Der Zielstring muss mindestens als `char name [9]` deklariert sein.

3.3.3 Funktionsaufrufe

int Snd (
const void * snd_buffer
, CM_INT32 send_len
)

Sendet die angegebenen Daten. Falls keine Conversation aktiv ist, werden alle dazu nötigen Aufrufe implizit erledigt.

Ergebnis:

`CUPIC_OK`	Aufruf war erfolgreich.
`CUPIC_ERROR`	Ein Fehler ist aufgetreten. Nähere Informationen können mit dem Aufruf *GetLastError()* abgefragt werden.

int SndLast (
const void * snd_buffer
, CM_INT32 send_len
)

Sendet die angegebenen Daten und gibt das Senderecht ab. Falls keine Conversation aktiv ist, werden alle dazu nötigen Aufrufe implizit erledigt.

Ergebnis:

`CUPIC_OK`	Der Aufruf war erfolgreich.
`CUPIC_ERROR`	Ein Fehler ist aufgetreten. Nähere Informationen können mit dem Aufruf *GetLastError()* abgefragt werden.

int Rcv (
void * rcv_buffer
, CM_INT32 buflen
, CM_INT32 * rcv_len
)

Empfängt eine Antwort.

Ergebnis:

`CUPIC_OK`	Der Aufruf war erfolgreich und die Conversation ist beendet.
`CUPIC_MORE_DATA`	Der Aufruf war erfolgreich, aber es ist nur ein Teil der Nachricht empfangen worden. Der Wert von `rcv_buffer` war für die vollständige Nachricht zu klein. `Rcv ()` muss wiederum aufgerufen werden, um die restlichen Daten zu erhalten.

`CUPIC_MORE_MSGS`	Der Aufruf war erfolgreich und die Nachricht ist vollständig eingelesen worden. Es können weitere Nachrichten empfangen werden. `Rcv ()` muss wiederum aufgerufen werden, um die nächste Nachricht zu erhalten.
`CUPIC_CONV_IS_OPEN`	Der Aufruf war erfolgreich, die letzte vollständige Nachricht wurde eingelesen und die Conversation ist noch offen. `Snd ()`, `SndLast ()`, `SndRcv ()` oder `Call ()` müssen aufgerufen werden, um die nächsten Daten zu senden.
`CUPIC_ERROR`	Ein Fehler ist aufgetreten. Nähere Informationen können mit dem Aufruf *GetLastError()* abgefragt werden.

int RcvMulti (
void * rcv_buffer
, CM_INT32 buflen
, CM_INT32 * rcv_len
)

Diese Funktion ermöglicht es, mehrere CPI-C-Nachrichten zu empfangen. Mehrere CPI-C-Nachrichten mit dem gleichen map name werden zu einer einzigen Nachricht zusammengefasst. Dies ist besonders nützlich, wenn mehrere Linemode-Nachrichten als eine einzige Nachricht empfangen werden sollen.

Ergebnis:

`CUPIC_OK`	Der Aufruf war erfolgreich und die Conversation ist geschlossen.
`CUPIC_MORE_DATA`	Der Aufruf war erfolgreich, aber es ist nur ein Teil der Nachricht empfangen worden. Der Wert von `rcv_buffer` war für die vollständige Nachricht zu klein. `Rcv ()` muss wiederum aufgerufen werden, um die restlichen Daten zu erhalten.
`CUPIC_MORE_MSGS`	Der Aufruf war erfolgreich, und eine vollständige Nachricht mit einem map name ist eingelesen worden. Es können weitere Nachrichten empfangen werden. `Rcv ()` muss wiederum aufgerufen werden, um die nächste Nachricht zu erhalten.
`CUPIC_CONV_IS_OPEN`	Der Aufruf war erfolgreich, die letzte vollständige Nachricht wurde eingelesen und die Conversation ist noch offen. `Snd ()`, `SndLast ()`, `SndRcv ()` oder `Call ()` müssen aufgerufen werden, um die nächsten Daten zu senden.
`CUPIC_ERROR`	Ein Fehler ist aufgetreten. Nähere Informationen können mit dem Aufruf *GetLastError()* abgefragt werden.

int SndRcv (
const void * send_buffer
, CM_INT32 send_len
, void * rcv_buffer
, CM_INT32 rcvbuf_len
, CM_INT32 * rcv_len
)

Sendet die angegebenen Daten und empfängt mindestens eine Antwort. Falls keine Conversation aktiv ist, werden alle dazu nötigen Aufrufe implizit erledigt. Dieser Aufruf ist eine Kombination von `Snd ()` und `Rcv ()`.

Ergebnis:

`CUPIC_OK`	Der Aufruf war erfolgreich und die Conversation ist beendet.
`CUPIC_MORE_DATA`	Der Aufruf war erfolgreich, aber es ist nur ein Teil der Nachricht empfangen worden. Der Wert von `rcv_buffer` war für die vollständige Nachricht zu klein. `Rcv ()` muss wiederum aufgerufen werden, um die restlichen Daten zu erhalten.
`CUPIC_MORE_MSGS`	Der Aufruf war erfolgreich und die Nachricht ist vollständig eingelesen worden. Es können weitere Nachrichten empfangen werden. `Rcv ()` muss wiederum aufgerufen werden, um die nächste Nachricht zu erhalten.
`CUPIC_CONV_IS_OPEN`	Der Aufruf war erfolgreich, die letzte vollständige Nachricht wurde eingelesen und die Conversation ist noch offen. `Snd ()`, `SndLast ()`, `SndRcv ()` oder `Call ()` müssen aufgerufen werden, um die nächsten Daten zu senden.
`CUPIC_ERROR`	Ein Fehler ist aufgetreten. Nähere Informationen können mit dem Aufruf *GetLastError()* abgefragt werden.

int Call (
const void * send_buffer
, CM_INT32 send_len
, void * rcv_buffer
, CM_INT32 rcvbuf_len
, CM_INT32 * rcv_len
)

Sendet die angegebenen Daten und empfängt mindestens eine Antwort. Falls keine Conversation aktiv ist, werden alle dazu nötigen Aufrufe implizit erledigt. Dieser Aufruf ist eine Kombination von `Snd ()` und `RcvMulti ()`.

Ergebnis:

`CUPIC_OK`	Der Aufruf war erfolgreich und die Conversation ist geschlossen.
`CUPIC_MORE_DATA`	Der Aufruf war erfolgreich, aber es ist nur ein Teil der Nachricht empfangen worden. Der Wert von `rcv_buffer` war für die vollständige Nachricht zu klein. `Rcv ()` muss wiederum aufgerufen werden, um die restlichen Daten zu erhalten.
`CUPIC_MORE_MSGS`	Der Aufruf war erfolgreich, und eine vollständige Nachricht mit einem map name ist eingelesen worden. Es können weitere Nachrichten empfangen werden. `Rcv ()` muss wiederum aufgerufen werden, um die nächste Nachricht zu erhalten.
`CUPIC_CONV_IS_OPEN`	Der Aufruf war erfolgreich, die letzte vollständige Nachricht wurde eingelesen und die Conversation ist noch offen. `Snd ()`, `SndLast ()`, `SndRcv ()` oder `Call ()` müssen aufgerufen werden, um die nächsten Daten zu senden.
`CUPIC_ERROR`	Ein Fehler ist aufgetreten. Nähere Informationen können mit dem Aufruf *GetLastError()* abgefragt werden.

int Restart (
void * rcv_buffer
, CM_INT32 rcvbuf_len
, CM_INT32 * rcv_len
)

Aktiviert den Wiederanlauf einer vorangegangenen Conversation und empfängt die Daten mit dem Aufruf `RcvMulti ()`.

Ergebnis:

`CUPIC_OK`	Der Aufruf war erfolgreich und die Conversation ist geschlossen.
`CUPIC_MORE_DATA`	Der Aufruf war erfolgreich, aber es ist nur ein Teil der Nachricht empfangen worden. Der Wert von `rcv_buffer` war für die vollständige Nachricht zu klein. `Rcv ()` muss wiederum aufgerufen werden, um die restlichen Daten zu erhalten.
`CUPIC_MORE_MSGS`	Der Aufruf war erfolgreich, und eine vollständige Nachricht mit einem map name ist eingelesen worden. Es können weitere Nachrichten empfangen werden. `Rcv ()` muss wiederum aufgerufen werden, um die nächste Nachricht zu erhalten.

`CUPIC_CONV_IS_OPEN`	Der Aufruf war erfolgreich, die letzte vollständige Nachricht wurde eingelesen und die Conversation ist noch offen. `Snd ()`, `SndLast ()`, `SndRcv ()` oder `Call ()` müssen aufgerufen werden, um die nächsten Daten zu senden.
`CUPIC_ERROR`	Ein Fehler ist aufgetreten. Nähere Informationen können mit dem Aufruf *GetLastError()* abgefragt werden.

void Reset()

Beendet die aktive Conversation und schließt die Transportverbindung.

BOOL Peek()

Test, ob Daten zum Empfang bereitstehen.

3.3.4 Public Diagnosefunktion

char * GetLastError ()

Gibt einen Textstring zurück, der den Fehler näher erklärt. Falls die Funktion *CUPIC_ERROR* zurückbekommt, dann wurde `Reset ()` bereits aufgerufen.

void GetLastError (
const char ** error_text
, CM_CALL_ID * c
, CM_RETCODE * rc
)

Gibt einen Textstring zurück, der den Fehler näher erklärt. Der letzte Aufruf (definiert als *CM_CALL_ID* in `upic.h`) und der letzte UPIC Returncode werden zurückgegeben.

char * GetDiagContext ()

Die Klasse CUpic schreibt alle ihre Aktionen in abdruckbarer Form in einen Diagnosekontext hinein. Diese Methode liefert diese Information, indem sie einen Pointer auf den entsprechenden Bereich zurückliefert.

void ResetDiagContext ()

Die Klasse CUpic schreibt alle ihre Aktionen in abdruckbarer Form in einen Diagnosekontext hinein. Diese Methode setzt den Inhalt des Diagnosekontext zurück.

3.4 Beispiel

```
#include "CUpic.h"
void main ( int argc, char * argv[] )
{
 char sbuf[1000];
 char rbuf[100000];
 CM_INT32 rcv_len;
 int rc;

 CUpic u;

 // Make a simple call based on configuration defaults memset (sbuf, '\0',
sizeof(sbuf));

 rc = u.Call (sbuf, strlen(sbuf), rbuf, sizeof(rbuf), &rcv_len);

 if ( rc == CUPIC_OK )
 {
  print ("%.*s", rcv_len, rbuf);
 }
 else
 {
  print ("%s", u. GetLastError() );
 }
 // Make a simple admin call overwriting configuration defaults
 CUpicLocAddr  l = CUpicLocAddr("its-me!", 4711);
 CUpicRemAddr  r = CUpicRemAddr("sample", "127.0.0.1", 30000);
 CUpicSecurity s = CUpicSecurity("admin");

 u. SetLocal (l);
 u. SetRemote (r);
 u. Set Security (s);
 u. SetTPName ("KDCINF");

 strcpy (sbuf, "STAT");

 rc = u. Call (sbuf, strlen(sbuf), rbuf, sizeof(rbuf), &rcv_len);

 if ( rc == CUPIC_OK )
 {
  printf ("%.*s", rcv_len, rbuf);
 }
 else
 {
  printf ("%s", u. GetLastError);
 }
}
```

4 CPI-C-Schnittstelle

Mit UPIC als Trägersystem können Sie CPI-C-Anwendungen, die auf dem lokalen Rechner ablaufen, mit UTM-Anwendungen koppeln, die im BS2000/OSD, auf Unix- oder Windows-Systemen laufen. Der vom Client angeforderte UTM-Service kann entweder die CPI-C- oder die KDCS-Schnittstelle von openUTM nutzen.

In diesem Kapitel finden Sie Informationen über:

– die allgemeine Struktur von CPI-C-Client-Programmen

– den Austausch von Nachrichten zwischen Client und Server

– die Konvertierung der ausgetauschten Daten bei heterogenen Kopplungen

– Programmierhinweise für die Kommunikation mit UTM-Einschritt- und UTM-Mehrschritt-Vorgängen

– Ablauf der Verschlüsselung

– die Programmierung von Client-Programmen, die parallel mit mehreren Services gekoppelt werden sollen (Multiple Conversations). Multiple Conversations sind nur möglich, wenn der Client auf einem System abläuft, das Multithreading unterstützt.

– die Security-Funktionen von openUTM, die beim Anschluss von UPIC-Client-Programmen genutzt werden können

– die CPI-C-Funktionen, die das Trägersystem UPIC unterstützt. Die einzelnen CPI-C-Funktionsaufrufe sind vollständig beschrieben (die CPI-C Specification von X/Open ist also nicht erforderlich).

Zunächst werden jedoch einige CPI-C-Begriffe erläutert, die in den folgenden Kapiteln verwendet werden.

4.1 CPI-C-Begriffe

Bei CPI-C gibt es die Begriffe Conversation, Conversation Characteristics und Side Information.

- Unter einer **Conversation** versteht man die Kommunikationsbeziehung, die ein CPI-C-Programm mit einem UTM-Service abwickelt.
- Die **Conversation Characteristics** beschreiben die aktuellen Parameter und Eigenschaften einer Conversation, siehe „Conversation Characteristics“ auf Seite 51.
- Die **Side Information** beschreibt beim Trägersystem UPIC im Wesentlichen die für eine Conversation notwendigen Adressierungsinformationen. Die für eine Conversation notwendigen Adressierungsinformationen können in der **Side Information-Datei (upicfile)** stehen.

Zustand einer Conversation

Der Zustand einer Conversation spiegelt die letzte Aktion dieser Conversation wider bzw. legt die erlaubten Folgeaktionen fest.

Wenn Sie ein Programm schreiben, das CPI-C-Aufrufe verwendet, müssen Sie darauf achten, dass im CPI-C-Programm und im UTM-Teilprogramm immer die passenden Aufrufe verwendet werden. Insbesondere kann immer nur der Partner Daten senden, der das Senderecht besitzt.

Eine Conversation kann sich beim Trägersystem UPIC in einem der folgenden Zustände befinden:

Zustand	**Beschreibung**
Start	Das Programm ist nicht beim Trägersystem UPIC angemeldet. (Vor *Enable_UTM_UPIC*-Aufruf oder nach *Disable_UTM_UPIC*-Aufruf)
Reset	Der *conversation_ID* ist keine Conversation zugeordnet.
Initialize	Der *Initialize_Conversation*-Aufruf wurde erfolgreich beendet und der Conversation wurde eine *conversation_ID* zugeordnet.
Send	Das Programm hat das Recht, Daten über die Conversation zu senden.
Receive	Das Programm kann Informationen über die Conversation empfangen.

Tabelle 2: Zustand einer Conversation

Eine Conversation befindet sich zu Beginn im Zustand "Reset" und nimmt danach verschiedene Folgezustände an, jeweils abhängig von den eigenen Aufrufen und den Informationen, die vom Partner-Programm empfangen wurden.

Eine besondere Rolle spielen die Zustände "Send" und "Receive", auf die im Abschnitt „Austausch von Nachrichten mit einem UTM-Service“ auf Seite 56 eingegangen wird. Eine Zustandstabelle finden Sie im Anhang auf Seite 359. Hier finden Sie die Zustandsänderungen einer CPI-C-Conversation in Abhängigkeit von den CPI-C-Aufrufen und ihren Ergebnissen.

UPIC überwacht den aktuellen Zustand einer Conversation. Falls die Synchronisation der beiden Seiten durch einen ungültigen Aufruf verletzt werden sollte, wird dieser Fehler mit dem Wert CM_PROGRAM_STATE_CHECK als Ergebnis des Aufrufs angezeigt.

Die X/Open CPI-C Specification definiert weitere Zustände, die aber beim Trägersystem UPIC nicht angenommen werden können.

Conversation Characteristics

Die Conversation Characteristics werden zusammen mit der Side Information einer Conversation in einem Kontrollblock verwaltet. Dieser Abschnitt beschreibt die für CPI-C mit Trägersystem UPIC relevanten Characteristics sowie die Werte, die ihnen beim Aufruf *Initialize_Conversation* zugewiesen werden. Die X/OPEN-Schnittstelle CPI-C enthält weitere, hier nicht aufgeführte Characteristics.

Es gibt drei Arten von Conversation Characteristics:

- fest vorgegebene
- veränderbare über CPI-C-Aufrufe
- UPIC-spezifische

Folgende Conversation Characteristics sind fest vorgegeben:

Conversation Characteristics	Initialisierungswert bei Initialize_Conversation
conversation_type	CM_MAPPED_CONVERSATION
return_control	CM_WHEN_SESSION_ALLOCATED
send_type	CM_BUFFER_DATA
sync_level	CM_NONE

Tabelle 3: Fest vorgegebene Conversation Characteristics

Folgende Conversation Characteristics sind über CPI-C-Aufrufe veränderbar:

Conversation Characteristics	**Initialisierungswert bei Initialize_Conversation**
deallocate_type	CM_DEALLOCATE_SYNC_LEVEL
partner_LU_name	Wert aus Side Information, abhängig vom Symbolic Destination Name
partner_LU_name_length	Länge von *partner_LU_name*
receive_type	CM_RECEIVE_AND_WAIT
security_new_password	leer
security_new_password_length	0
security_password	Leerzeichen
security_password_length	0
security_type	CM_SECURITY_NONE
security_user_ID	Leerzeichen
security_user_ID_length	0
TP_name	Wert aus Side Information abhängig vom Symbolic Destination Name
TP_name_length	Länge von *TP_name*

Tabelle 4: Veränderbare Conversation Characteristics

Folgende Conversation Characteristics sind UPIC-spezifisch und veränderbar, dabei wird zwischen den Charakteristics für eine Partner-Anwendung und den Werten für eine lokale Anwendung unterschieden:

Conversation Characteristics	**Initialisierungswert bei Initialize_Conversation**
CHARACTER_CONVERTION	CM_NO_CHARACTER_CONVERTION
CLIENT_CONTEXT	leer
COMMUNICATION_PROTOCOL	CM_COMMUNICATION_PROTOCOL_40
ENCRYPTION-LEVEL	0
PORT	102
T-SEL	wird aus *partner_LU_name* abgeleitet
T-SEL-FORMAT	wird aus *partner_LU_name* abgeleitet
HOSTNAME	wird aus *partner_LU_name* abgeleitet
IP-ADDRESS	wird nicht initialisiert
RSA-KEY	wird von der UTM-Anwendung vergeben
SECONDARY_RETURN_CODE	CM_RETURN_TYPE_SECONDARY
TRANSACTION_STATE	leer

Tabelle 5: UPIC-spezifische Conversation Characteristics für ferne Anwendungen

Werte für lokale Anwendung	**Initialisierungswert bei Enable_UTM_UPIC**
PORT	102
T-SEL	wird aus dem lokalen Anwendungsnamen abgeleitet
T-SEL-FORMAT	wird aus dem lokalen Anwendungsnamen abgeleitet

Tabelle 6: UPIC-spezifische Werte für lokale Anwendungen

Die Bedeutung der Characteristics und lokalen Werte wird nicht näher erklärt. Diese Aufzählung wird nur vorgenommen, um einen Vergleich der von UPIC zur Verfügung gestellten Schnittstelle CPI-C mit der X/Open-Schnittstelle CPI-C hinsichtlich der Conversation Characteristics zu ermöglichen. Eine detaillierte Erklärung finden Sie in der X/Open Spezifikation „CPI-C Specification Version 2“.

Side Information

Da die Adressierungsinformationen abhängig sind von der jeweiligen Konfiguration, verwenden CPI-C-Anwendungen folgende symbolische Namen für die Adressierung:

- Symbolic Destination Name
  Der *Symbolic Destination Name* adressiert den Kommunikationspartner. Hinter dem *Symbolic Destination Name* verbergen sich die folgenden Komponenten:
  - *partner_LU_name*
    Sie adressiert die UTM-Partner-Anwendung und kann im Programm mit *Set_Partner_LU-name* überschrieben werden.
  - *TP_name*
    Sie adressiert den UTM-Service innerhalb der UTM-Partner-Anwendung. *TP_name* ist ein Transaktionscode und kann vom Programm mit *Set_TP_Name* überschrieben werden, z.B. *TP_name*=KDCDISP für den Wiederanlauf.

    Der durch diesen Transaktionscode adressierte UTM-Service wird gestartet, sobald das Programm den ersten *Receive*-Aufruf oder einen *Prepare_To_Receive*-Aufruf abgesetzt hat.
  - *Schlüsselwörter*
    Mit verschiedenen Schlüsselwörtern können weitere UPIC-spezifische Conversation Characteristics gesetzt werden. Ein Programm kann diese Characteristics mit den entsprechenden CPI-C-Aufrufen (z.B. *Set_Encryption_Level*) überschreiben.

  Der *Symbolic Destination Name* wird über die `upicfile` mit der „realen" Adressierung (*partner_LU_name*, *TP_name*) verknüpft. *partner_LU_name* und *TP_name* und die Schlüsselwörter gehören zu den Conversation Characteristics, die unten beschrieben werden.

- *local_name*
  Der *local_name* vergibt für die eigene Anwendung den lokalen Anwendungsnamen. Für den *local_name* kann in der `upicfile` ein symbolischer Name vergeben werden. Mit Schlüsselwörtern können UPIC-lokale Werte gesetzt werden. Dadurch wird der Name, den das Programm vergibt, unabhängig vom Namen, der in der TNS- bzw. UTM-Generierung verwendet wird. Ein Programm kann diese Characteristics mit den entsprechenden CPI-C-Aufrufen (z.B. *Specify_Local_Tsel*) überschreiben.

Wie die `upicfile` erstellt wird und wie die Einträge mit der TNS- und UTM-Generierung zusammenhängen, ist in Abschnitt „Abstimmung mit der Partnerkonfiguration" auf Seite 315ff beschrieben.
Wenn eine `upicfile` verwendet wird, so hat dies den Vorteil, dass TNS- und UTM-Generierung geändert werden können (z.B. die UTM-Server-Anwendung auf einen anderen Rechner umziehen), ohne dass die Client-Programme geändert werden müssen.

4.2 Allgemeiner Aufbau einer CPI-C-Anwendung

Eine CPI-C-Anwendung ist ein Hauptprogramm und enthält in der Regel:

- Bedienung einer Schnittstelle zu einem Präsentationssystem
- interne Verarbeitungsroutinen (evtl. Bedienung weiterer Schnittstellen)
- Bedienung der CPI-C-Schnittstelle (zu einer UTM-Anwendung)
- Überblick über spezielle CPI-C- und UTM-Funktionen, die die Clients über UPIC nutzen können

Reihenfolge der Aufrufe in einer CPI-C-Anwendung

Für die im Abschnitt „CPI-C-Aufrufe bei UPIC“ auf Seite 96ff beschriebenen Schnittstellen-Aufrufe gelten folgende Regeln:

1. Der erste CPI-C-Funktionsaufruf in Ihrem Programm muss ein *Enable_UTM_UPIC*-Aufruf, der letzte muss ein *Disable_UTM_UPIC*-Aufruf sein. Zwischen diesen beiden Aufrufen können Sie andere CPI-C-Aufrufe gemäß den nachfolgend beschriebenen Regeln beliebig oft wiederholen. Durch *Enable_UTM_UPIC* wird die Ablaufumgebung für den Client gestartet.

2. Nach dem *Enable_UTM_UPIC*-Aufruf können Sie mit den *Specify_...*-Aufrufen die UPIC-spezifischen Werte der lokalen Anwendung verändern.

3. Sie müssen mit *Initialize_Conversation* die Characteristics für die Conversation initialisieren. Die Characteristics sind im „Conversation Characteristics“ auf Seite 51 beschrieben.

4. Nach dem Initialisieren können Sie mit den *Set_...* - Aufrufen verschiedene Conversation Characteristics setzen oder ändern (siehe Seite 52; veränderbare Characteristics).

5. Mit dem *Allocate*-Aufruf müssen Sie die Conversation einrichten.

6. Nach einem *Allocate* können Sie mit den Aufrufen *Send_Data*, *Send_Mapped_Data* sowie *Prepare_To_Receive*, *Receive* und *Receive_Mapped_Data* die Verarbeitung durchführen. Nach dem *Allocate* muss jedoch zunächst ein *Send_Data-* oder *Send_Mapped_Data*-Aufruf erfolgen, bevor das Programm mit *Receive* oder *Receive_Mapped_Data* Daten vom UTM-Server empfangen kann. Mehr Informationen zu den *Send*- und *Receive*-Aufrufen finden Sie im Abschnitt „Austausch von Nachrichten mit einem UTM-Service“ auf Seite 56.

Soll ein CPI-C-Programm nacheinander mehrere Conversations unterhalten, dann empfiehlt es sich aus Performancegründen, in einer CPI-C-Anwendung jeweils nur einen *Enable_UTM_UPIC-* und einen *Disable_UTM_UPIC*-Aufruf abzusetzen, d.h. nicht vor jedem *Initialize_Conversation* einen *Enable*-Aufruf und nach jedem Beenden der Conversation einen *Disable*-Aufruf.

Soll ein CPI-C-Programm gleichzeitig mehrere Conversations unterhalten, dann muss für jede dieser Conversations vor dem *Initialize_Conversation* ein *Enable_UTM_UPIC*-Aufruf erfolgen. Alle CPI-C-Aufrufe, die zu einer Conversation gehören, müssen in demselben Thread erfolgen. Siehe dazu Abschnitt „Multiple Conversations“ auf Seite 89.

4.3 Austausch von Nachrichten mit einem UTM-Service

Nach dem Einrichten einer Conversation zwischen einem Client und einem UTM-Service muss der Client zur Steuerung des Services Nachrichten an den UTM-Service übergeben. Der Service schickt dem Client das Ergebnis der Bearbeitung in Form einer Nachricht zurück. Dabei ist jedoch zu beachten, dass auf einer Conversation zu einem Zeitpunkt immer nur eine Seite (Client oder Service) Daten senden darf. Man sagt, diese Seite der Conversation hat das Senderecht. Das Senderecht muss explizit an die andere Seite der Conversation übertragen werden, damit der Partner senden kann.

In diesem Abschnitt ist beschrieben

- wie der Nachrichtenaustausch abläuft,
- was Sie beim Programmieren einer Client-Anwendung beachten müssen und
- welche Funktionen für den Nachrichtenaustausch zur Verfügung stehen.

Im Abschnitt „Kommunikation mit dem openUTM-Server“ auf Seite 71 finden Sie detaillierte Beispiele für die Kommunikation zwischen Client und UTM-Server-Anwendung. Dort wird der Programmablauf auf Client-Seite und der auf Server-Seite (Schnittstelle KDCS) gegenübergestellt.

4.3.1 Nachricht senden und UTM-Service starten

Im folgenden Bild sind die Abläufe im Client-Programm dargestellt, durch die der Client den Service in der UTM-Server-Anwendung startet und eine Nachricht an den Service übergibt.

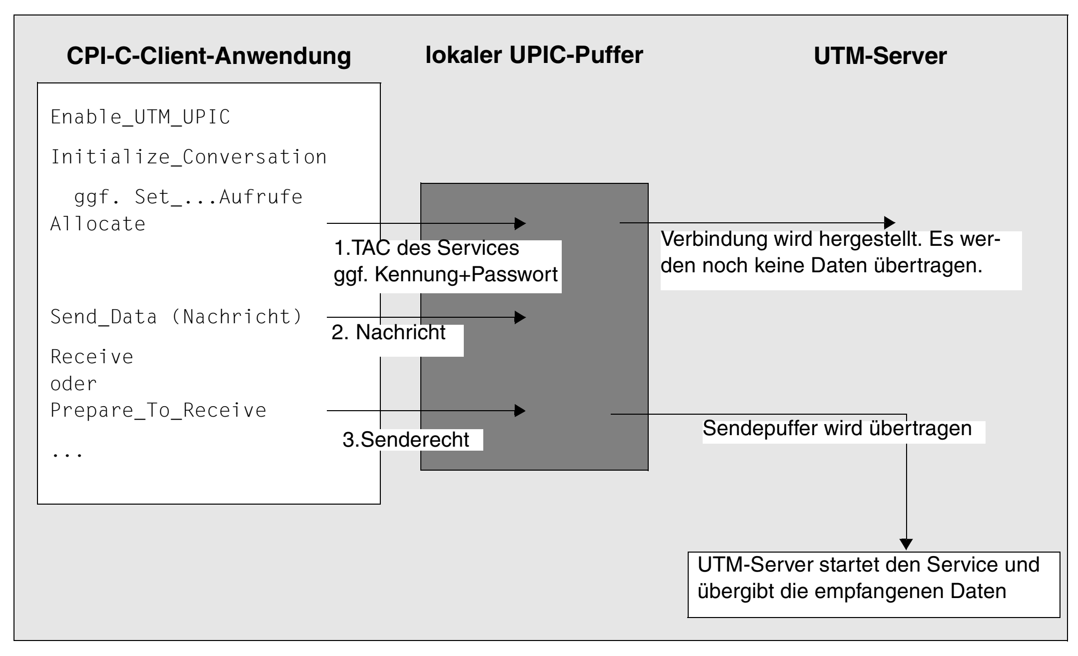


Bild 6: Client startet einen Service in der UTM-Partner-Anwendung

Erläuterungen zum Bild

1. Nach dem *Allocate*-Aufruf ist die Conversation „eingerichtet“ und eine Verbindung zum UTM-Server hergestellt. Der UTM-Service ist jedoch noch nicht gestartet. UPIC verwaltet jetzt einen internen Puffer, in den die Daten der Conversation geschrieben werden.

2. Nach dem *Allocate*-Aufruf befindet sich der Client im Zustand „Send“; er hat das Senderecht auf der Conversation und muss jetzt eine Nachricht für den adressierten Service (*TP_Name*) an UPIC übergeben. Die Nachricht muss die Eingabedaten enthalten, die der Service bearbeiten soll. Dazu stehen dem Client folgende *Send*-Aufrufe zur Verfügung:

   Send_Data
   Send_Mapped_Data

Nach dem *Allocate*-Aufruf dürfen Sie mit *Set_...*-Aufrufen noch die Conversation Characteristic *receive_type* und die Werte für den Receive-Timer und die Funktionstaste ändern.

Send_Mapped_Data unterscheidet sich vom *Send_Data*-Aufruf dadurch, dass neben der Nachricht auch Formatnamen an den Server übertragen werden. Entsprechend kann der Client mit *Receive_Mapped_Data* Daten zusammen mit den Formatnamen vom Service empfangen. Siehe dazu Abschnitt „Formate senden und empfangen“ auf Seite 62.

Durch den *Send*-Aufruf werden die Daten von UPIC in einen lokalen Sendepuffer geschrieben, der dem UTM-Service am lokalen System eindeutig zugeordnet ist.
Der Client kann zur Übergabe der Nachricht mehrere *Send*-Aufrufe absetzen.

Benötigt der UTM-Service zur Bearbeitung der Anforderung keine Daten, dann muss der Client eine leere Nachricht an den Server senden.

3. Nachdem der Client die Nachricht vollständig an UPIC übergeben hat, muss er das Senderecht an den Server übergeben, indem er in den Zustand "Receive" wechselt. Dazu stehen folgende CPI-C-Aufrufe zur Verfügung:

   Receive
   Receive_Mapped_Data
   Prepare_To_Receive

   Erst jetzt überträgt UPIC den letzten Teil des Sendepuffers zusammen mit dem Senderecht an den UTM-Service. Das zugehörige Teilprogramm der UTM-Server-Anwendung wird gestartet.

   Wenn Sie einen *Receive*-Aufruf nutzen, um das Senderecht an die UTM-Anwendung zu übertragen, dann überträgt der Client das Senderecht und wartet danach im *Receive* auf die Antwort vom Service (blockierender Receive; siehe Abschnitt „Nachricht empfangen, blockierender und nicht-blockierender Receive“ auf Seite 59).

   Der Aufruf *Prepare_To_Receive* bewirkt, dass der lokale UPIC-Sendepuffer sofort zusammen mit dem Senderecht an den Server übertragen wird. Der Client wechselt in den Zustand "Receive", empfängt jedoch noch keine Daten. Zum Empfang der Antwort vom UTM-Service muss der Client *Receive* oder *Receive_Mapped_Data* aufrufen. Vor diesem *Receive*-Aufruf kann der Client jedoch weitere (lokale) Verarbeitungsschritte, die die CPI-C-Schnittstelle nicht nutzen, durchführen. Da die Conversation sich im Zustand "Receive" befindet, sind zwischen *Prepare_To_Receive* und dem *Receive-* bzw. *Receive_Mapped_Data*-Aufruf nur die CPI-C-Aufrufe *Set_Receive_Type, Set_Receive_Timer* und *Set_Function_Key* erlaubt.
   Prepare_To_Receive bietet sich an, wenn Sie einen „langlaufenden“ Service starten, bei dem nicht unmittelbar mit einer Antwort zu rechnen ist, z.B. Services mit mehreren Datenbankzugriffen oder mit verteilter Transaktionsverarbeitung zwischen der UTM-Partner-Anwendung und anderen Server-Anwendungen. Das Client-Programm und der Prozess sind dann nicht für die gesamte Bearbeitungszeit blockiert.

4.3.2 Nachricht empfangen, blockierender und nicht-blockierender Receive

Der UTM-Service übergibt seine Ergebnisse in Form einer Nachricht bzw. mehrerer Teilnachrichten an den Client. Dabei kann es sich auch um eine leere Nachricht handeln. Darüber hinaus überträgt der UTM-Server entweder das Senderecht an den Client oder er beendet die Conversation. Die Nachricht vom UTM-Service wird von UPIC empfangen und lokal in einem Empfangspuffer abgelegt. Der Client kann die Nachricht bei Bedarf aus dem Empfangspuffer übernehmen. Dazu stehen ihm folgende *Receive*-Aufrufe zur Verfügung:

Receive
Receive_Mapped_Data

Jede Teilnachricht vom UTM-Service (jedes MPUT NT/NE) muss mit einem eigenen *Receive*-Aufruf empfangen werden. Das Senderecht erhält der Client, wenn beim *Receive*-Aufruf im Feld *status_received* der Wert CM_SEND_RECEIVED steht.

Wenn der UTM-Service sich beendet (PEND FI), wird die Conversation vom Server beendet. Dem Client wird beim *Receive* der Returncode CM_DEALLOCATE_NORMAL zurückgeliefert und die Conversation geht in den Zustand "Reset" über.

i Ein CPI-C-Programm muss immer mindestens einen *Receive*-Aufruf absetzen, d.h. *Send*-Aufrufe ohne nachfolgenden *Receive*-Aufruf sind nicht erlaubt.

Dem folgenden Bild können Sie die Abläufe beim Empfangen von Nachrichten im Client-Programm entnehmen.

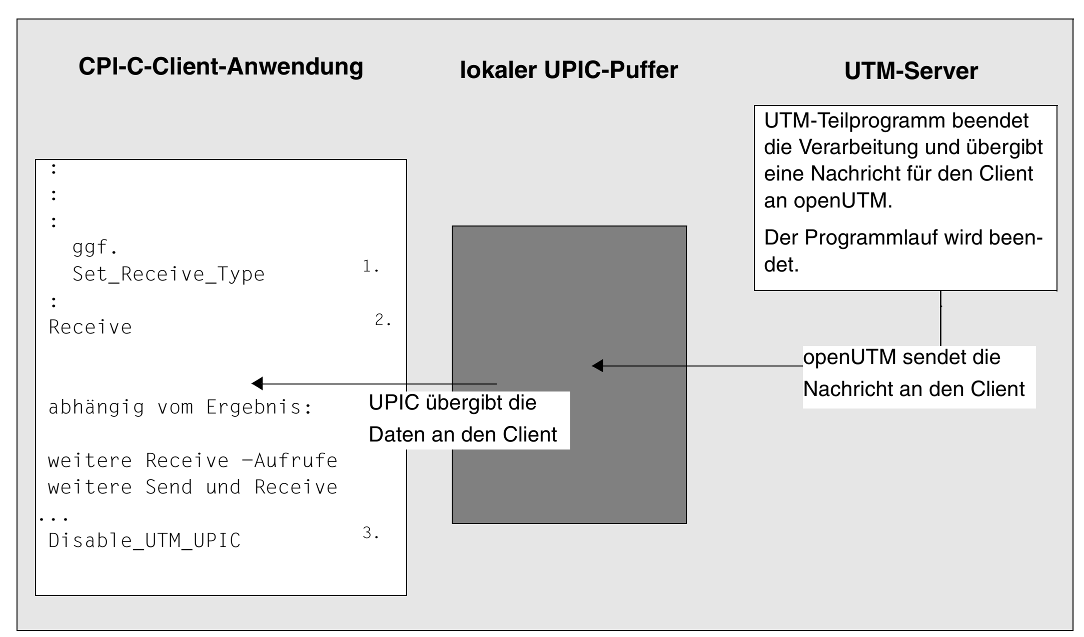


Bild 7: Client empfängt eine Nachricht vom Service, die Conversation wird abgebaut

Erläuterungen zum Bild

1. Mit dem Aufruf *Set_Receive_Type* können Sie festlegen, ob der Empfang der Daten blockierend oder nicht-blockierend erfolgen soll.
   Ob ein *Receive*-Aufruf blockierend oder nicht-blockierend bearbeitet wird, ist abhängig vom Wert der Conversation Characteristic *receive_type*. Nach dem Initialisieren der Conversation Characteristics mit dem Aufruf *Initialize_Conversation* ist der blockierende *Receive* für die Conversation eingestellt. Mit Hilfe des Aufrufs *Set_Receive_Type* können Sie diese Voreinstellung ändern.

   Beim **blockierenden** *Receive*-Aufruf (*receive_type*=CM_RECEIVE_AND_WAIT) wartet das Client-Programm solange im *Receive* bzw. *Receive_Mapped_Data*, bis Daten vom Server für die Conversation eingetroffen sind, oder der Aufruf durch einen Timer unterbrochen wird. Erst dann wird die Kontrolle an das Client-Programm zurückgegeben, der Programmlauf kann fortgesetzt werden.

Wenn Sie mit dem blockierenden Receive arbeiten, dann sollten Sie sicherstellen, dass das Programm nicht „endlos“ wartet. Dazu sollten in der UTM-Server-Anwendung entsprechende Timer gesetzt werden (siehe openUTM-Handbuch „Anwendungen administrieren“ und das openUTM-Handbuch „Anwendungen generieren“). Auf seiten des Client kann mit *Set_Receive_Timer* ein Timeout-Timer für den blockierenden *Receive* gesetzt werden.

Beim **nicht-blockierenden** *Receive*-Aufruf (*receive_type*=CM_RECEIVE_IMMEDIATE) erhält das Programm sofort die Kontrolle zurück. Liegen zum Zeitpunkt des Aufrufs Daten vom Service vor, dann werden sie an das Programm übergeben. Liegen zum Zeitpunkt des Aufrufs keine Daten vor, dann liefert der Aufruf den Returncode CM_UNSUCCESSFUL.

Die Characteristic *receive_type* kann innerhalb der Conversation beliebig oft geändert werden. Bei jedem *Receive* gilt die Einstellung, die durch den letzten *Set_Receive_Type*-Aufruf vor dem *Receive* festgelegt wurde.

Upic-Local:
Bei der lokalen Anbindung über UPIC-Local werden der nicht-blockierende *Receive* und der Aufruf *Set_Receive_Type* nicht unterstützt.

2. Mit dem *Receive-* bzw. *Receive_Mapped_Data*-Aufruf liest der Client die Daten aus dem Empfangspuffer. Liegen Daten vor, dann übergibt der *Receive*-Aufruf die Daten direkt an das Client-Programm. Der weitere Verlauf des Client-Programms ist abhängig vom Ergebnis des *Receive*-Aufrufs (Felder *data_received, status_received, return_code*). Folgende Ergebnisse können auftreten:

   – Hat das Programm die Nachricht mit dem *Receive*-Aufruf vollständig gelesen (*data_received*=CM_COMPLETE_DATA_RECEIVED) und der UTM-Service die Conversation beendet (PEND FI aufgerufen), dann geht das Programm in den Status "Reset" über. Es kann jetzt eine neue Conversation aufbauen oder sich mit *Disable_UTM_UPIC* bei UPIC abmelden.

   – Das Programm hat noch nicht alle Teilnachrichten gelesen, die vom Service empfangen wurden. *Receive*-Aufrufe müssen solange abgesetzt werden, bis *data_received* den Wert CM_COMPLETE_DATA_RECEIVED annimmt. Pro Teilnachricht, die der Service sendet (MPUT NT), muss ein *Receive*-Aufruf abgesetzt werden.

   – Das Programm hat die vollständige Nachricht vom Service gelesen, der Service überträgt dem Client das Senderecht (*status_received*=CM_SEND_RECEIVED). Dann muss der Client als nächstes mindestens einen *Send*-Aufruf und dann erneut *Receive*-Aufrufe absetzen. Der UTM-Service ist in diesem Fall ein Mehrschritt-Vorgang (das Teilprogramm hat sich mit PEND KP beendet).

3. Nachdem die letzte Conversation beendet ist, ruft das Client-Programm *Disable_UTM_UPIC* auf, um sich bei UPIC abzumelden.

4.3.3 Formate senden und empfangen

Ein CPI-C-Client, der das Trägersystem UPIC nutzt, kann zusammen mit einer Benutzernachricht Formatnamen an einen UTM-Service senden und Formatnamen von einem UTM-Service empfangen.

Die mit der Benutzernachricht übergebenen Formatnamen können zur Beschreibung des Datenformats der Benutzerdaten dienen. Die Benutzerdaten und Formatnamen werden zwischen Client und Server transparent übertragen, d.h. sie können beliebige Bit-Kombinationen enthalten, die vom Empfänger der Nachricht interpretiert werden müssen. Dabei wird die Benutzernachricht nicht von einem Formatierungssystem anhand des Formatnamens bearbeitet.

Die zwischen UPIC und openUTM ausgetauschten Formatnamen sind in der Regel frei wählbar, ebenso wie die Struktur. Die Strukturinformation ist dann von Bedeutung, wenn für Terminals geschriebene Programme auch mit UPIC-Clients kommunizieren sollen. In diesem Fall spielt das Formatkennzeichen eine Rolle, das aus einem Präfix (-, +, # oder *) und dem eigentlichen Formatnamen besteht.

UPIC-Client und UTM-Programm verwenden die Formatnamen, die in der UTM-Anwendung definiert sind, um die Strukturierungsmerkmale einer Nachricht festzulegen. Zu jedem Formatkennzeichen, das die UTM-Anwendung kennt, existiert in der UTM-Anwendung eine Datenstruktur (Adressierungshilfe). Durch diese Funktion kann ein UPIC-Client auch TM-Anwendungen aufrufen, die mit Terminals über Formate kommunizieren. Dazu muss das Client-Programm das Formatkennzeichen übergeben, dass das UTM-Programm erwartet. Die Benutzernachricht muss dann entsprechend dem Formatkennzeichen aufgebaut sein.

Analog übergibt die UTM-Server-Anwendung beim Senden von Formatdaten das Formatkennzeichen an das Client-Programm, das die Struktur des Nachrichtenbereichs beschreibt.

CPI-C-Aufrufe zum Austausch von Formatdaten

Da die CPI-C-Schnittstelle kein eigenes Konzept zur Übergabe von Formatnamen an der Schnittstelle hat, benutzt UPIC die Funktionen

Send_Mapped_Data
Receive_Mapped_Data

um Nachrichten zusammen mit Formatnamen zu senden bzw. zu empfangen.

Zum Senden von Formatdaten an die UTM-Server-Anwendung muss das Client-Programm *Send_Mapped_Data* aufrufen. Im Feld *map_name* des Aufrufs übergibt der Client das Formatkennzeichen als Strukturierungsmerkmal der Nachricht, die an die UTM-Server-Anwendung gesendet werden soll.
Die Nachricht muss entsprechend des in der Server-Anwendung definierten Formats strukturiert sein. *Send_Mapped_Data* ist im Abschnitt „Send_Mapped_Data - Daten und Formatkennzeichen senden“ auf Seite 181 beschrieben.

Liefert der UTM-Service ein Format zurück, dann muss das Client-Programm *Receive_Mapped_Data* aufrufen, um die Nachricht zusammen mit dem Formatkennzeichen vom UTM-Service zu empfangen. Im Feld *map_name* übergibt UPIC das Formatkennzeichen, das der Server zur Strukturierung der Nachricht verwendet hat. Im Client-Programm muss die Nachricht entsprechend der vom UTM-Service verwendeten Strukturierung interpretiert werden. *Receive_Mapped_Data* ist im Abschnitt „Receive_Mapped_Data - Daten und Formatkennzeichen von einem UTM-Service empfangen“ auf Seite 166 beschrieben.

Sollen mehrere Teilformate an einen UTM-Service gesendet werden, dann muss das Client-Programm für jedes Teilformat einen eigenen *Send_Mapped_Data*-Aufruf absetzen. Der UTM-Service liest jedes Teilformat mit einem eigenen MGET NT-Aufruf.

Entsprechend gilt: besteht eine Nachricht vom UTM-Service aus mehreren Teilformaten, dann muss das Client-Programm für jedes Teilformat einen *Receive_Mapped_Data*-Aufruf absetzen.

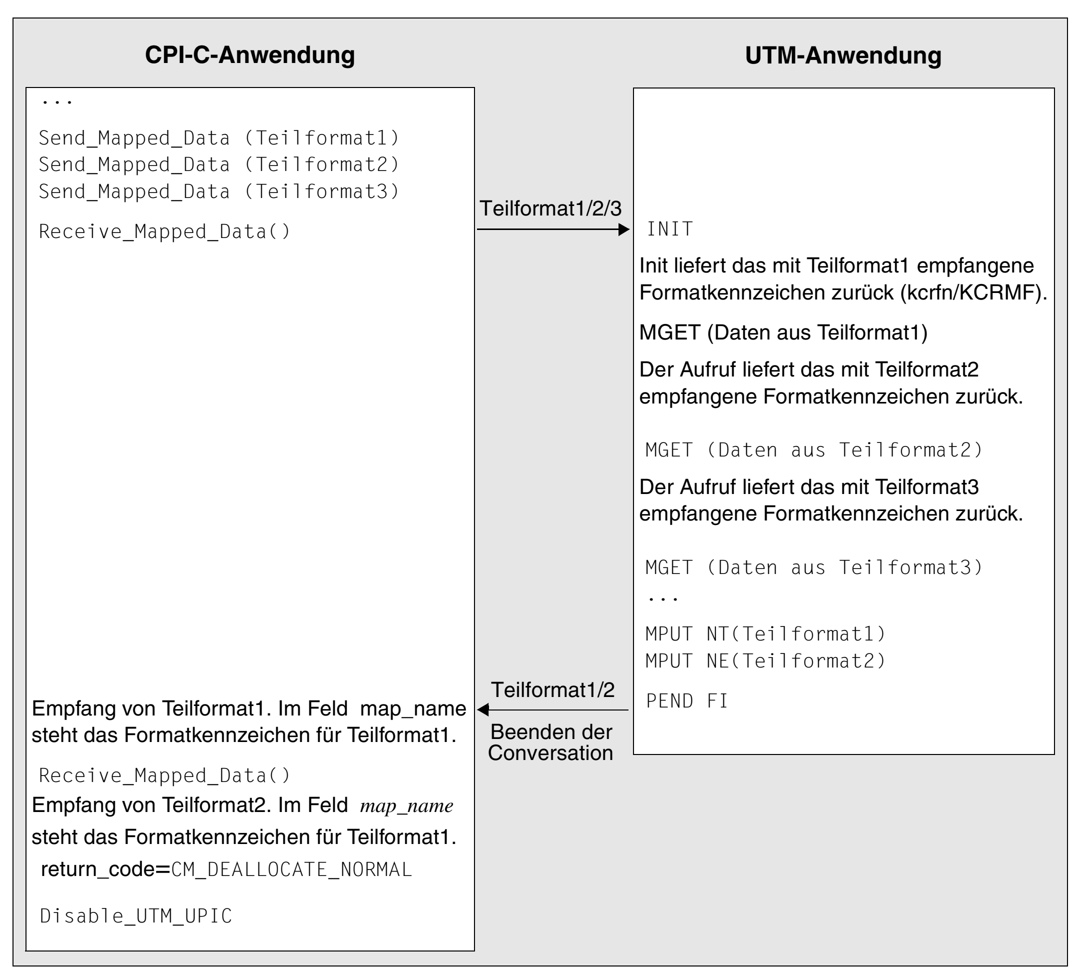


Bild 8: Austausch von Formaten

Detaillierte Informationen zum Arbeiten mit Formaten in einer UTM-Server-Anwendung finden Sie im openUTM-Handbuch „Anwendungen programmieren mit KDCS“.

openUTM-Formatkennzeichen und -Formattypen

Die zwischen einem UPIC-Client-Programm und einem UTM-Teilprogramm ausgetauschten Formatnamen können aus bis zu acht beliebigen Zeichen bestehen. Wichtig ist, dass beide Kommunikationspartner über Struktur und Bedeutung der mit dem Formatnamen übertragenen Benutzerdaten einig sind.

Wenn ein Client-Programm ein UTM-Teilprogramm aufruft, das auch mit Terminals über Formatkennzeichen kommuniziert, muss das Formatkennzeichen den Regeln der von openUTM unterstützten Formatierungssysteme entsprechen. Diese Formatkennzeichen bestehen aus:

- einem ein Byte langen Präfix, das den Typ des Formats angibt (mögliche Werte sind „*“, „+“, „#“ und mit openUTM auf BS2000/OSD zusätzlich „-“)
- einem bis zu 7 Zeichen langen Formatnamen.

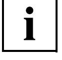
Die Formattypen unterscheiden sich wie folgt:

*Formate:
Die Anzeigeattribute der Formatfelder können nicht durch ein UTM-Teilprogramm geändert werden. Es wird nur der Inhalt der Datenfelder übertragen.

+Formate und #Formate:
Ein UTM-Teilprogramm kann die Anzeigeattribute der Datenfelder bzw. globale Attribute ändern. Den Datenfeldern sind deshalb Attributfelder bzw. -blöcke zugeordnet. Wird ein +Format oder #Format ausgetauscht, dann muss das Client-Programm diese Attributfelder berücksichtigen.

-Formate
sind nur in UTM-Anwendungen auf BS2000/OSD möglich. Es sind Formate, die mit dem Event-Exit FORMAT erstellt werden.

Näheres zu Formatkennzeichen und -typen finden Sie im openUTM-Handbuch „Anwendungen programmieren mit KDCS“.



Wenn ein UTM-Teilprogramm nur mit UPIC-Client Teilprogrammen kommuniziert, dann brauchen die Regeln für Formatkennzeichen nicht beachtet werden. Formatierungssysteme spielen bei dieser Form der Kommunikation keine Rolle.

4.3.4 UTM-Funktionstasten

In einer UTM-Server-Anwendung können Funktionstasten generiert werden (F1, F2, ...F24 und in BS2000/OSD zusätzlich K1 bis K14). Jeder Funktionstaste kann per UTM-Generierung eine bestimmte Funktion zugeordnet werden, die openUTM ausführt, wenn die Funktionstaste betätigt wird.

Ein CPI-C-Client-Programm kann Funktionstasten in einer UTM-Server-Anwendung auslösen.

Zum „Betätigen einer UTM-Funktionstaste“ steht der Funktionsaufruf *Set_Function_Key* zur Verfügung. *Set_Function_Key* ist eine UPIC-spezifische Funktion, sie gehört nicht zum Funktionsumfang der X/Open-CPI-C-Schnittstelle.

Mit *Set_Function_Key* gibt das Client-Programm die Funktionstaste an, die in der UTM-Server-Anwendung ausgelöst werden soll.

Der dieser Funktionstaste zugeordnete Returncode wird dem UTM-Service von openUTM beim ersten MGET-Aufruf übergeben (Feld KCRCCC). Über den Returncode kann der Teilprogrammlauf des UTM-Services gesteuert werden (z.B. ein bestimmter Folge-Tac gestartet werden). Zum Lesen der Nachricht vom Client, die dieser mit *Send_Mapped_Data* gesendet hat, muss ein zweiter MGET-Aufruf erfolgen.

Der Aufruf von *Set_Function_Key* ist nur im Zustand "Send" und "Receive" erlaubt. Die Funktionstaste wird zusammen mit den Daten des folgenden *Send*-Aufrufs an den Service übertragen.

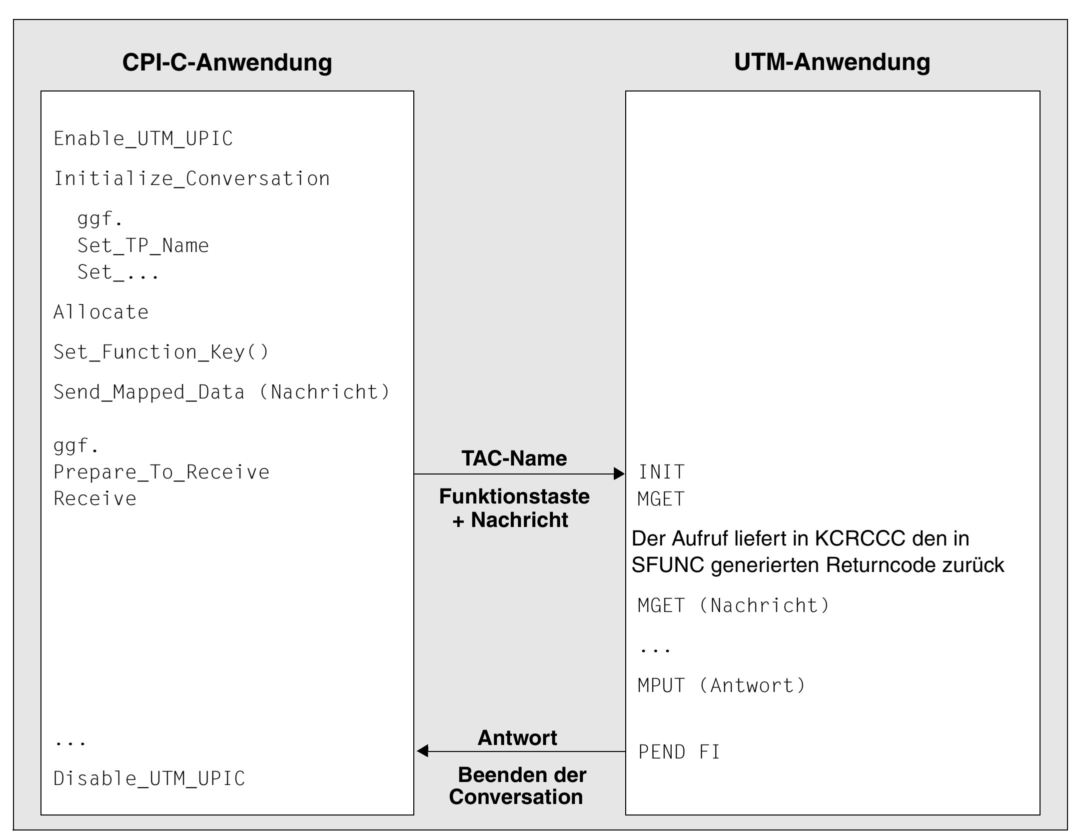


Bild 9: Betätigen einer Funktionstaste in der UTM-Server-Anwendung

4.3.5 Cursor-Position

Wenn in einem UTM-Teilprogramm in einem Dialogschritt eine Formatausgabe vorgesehen ist und mittels des Aufrufs KDCSCUR der Cursor auf ein Feld gesetzt werden soll, so wird diese Information an UPIC übertragen. openUTM bildet aus Differenz der Adresse des angegebenen Feldes und der Startadresse des Formats einen Offset. Dieser Offset wird an den UPIC-Client übertragen und kann mit dem Aufruf *Extract_Cursor_Offset* abgefragt werden.

Der Aufruf *Extract_Cursor_Offset* liefert einen Rückgabewert. Ist dieser Wert 0, wurde KDCSCUR im UTM-Teilprogramm nicht aufgerufen, es sei denn, dass der Cursor an den Beginn des Formates gesetzt werden soll und damit der Aufruf tatsächlich den Offset 0 liefert. Wenn KDCSCUR im UTM-Teilprogramm aufgerufen wurde, liefert *Extract_Cursor_Offset* die Cursor-Adresse im Format relativ zum Anfang des Nachrichtenbereichs als ganze Zahl.

4.3.6 Code-Konvertierung

Bei einer heterogenen Kopplung zu einer UTM-Server-Anwendung ist zu beachten, dass in den Systemen von Client und Server u. U. mit verschiedenen Codes (ASCII, EBCDIC) gearbeitet wird, z.B.:

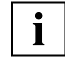


– Eine Client-Anwendung, die auf einem Unix- oder Windows-System abläuft, kommuniziert mit einer UTM-Server-Anwendung auf einem BS2000/OSD-System.



– Eine Client-Anwendung, die auf einem BS2000/OSD-System abläuft, kommuniziert mit einer UTM-Server-Anwendung auf einem Unix- oder Windows-System.

Unix- und Windows-Systeme verwenden einen ASCII-Code, BS2000/OSD-Systeme einen EBCDIC-Code. Bei der Kopplung eines ASCII-Systems mit einem EBCDIC-System können Nachrichten, die aus abdruckbaren Zeichen (7-Bit ASCII-Zeichensatz) bestehen, z.B. für die Ausgabe konvertiert werden. Reine Binärdaten dürfen nicht konvertiert werden. Die Konvertierung kann auf der Client-Seite oder auf der Server-Seite erfolgen. Sie müssen darauf achten, dass die Konvertierung nur einmal erfolgt.



Code-Konvertierung für UPIC-Clients kann bei openUTM nicht generiert werden (Parameter MAP für PTERM und TPOOL darf bei UPIC-Clients nur den Wert USER haben). Die Konvertierung auf Server-Seite muss daher im Teilprogramm durch den Anwender erfolgen.

Soll die Konvertierung im Client erfolgen, dann stehen beim Trägersystem UPIC zwei Möglichkeiten zur Verfügung:

- Die CPI-C-Aufrufe *Convert_Incoming* und *Convert_Outgoing*
  In diesem Fall werden die Daten vom Programm konvertiert. Mit *Convert_Incoming* können Sie eine empfangene Nachricht in den lokal verwendeten Code konvertieren (siehe Abschnitt „Convert_Incoming - Konvertieren vom Code des Senders in lokalen Code“ auf Seite 103). Mit *Convert_Outgoing* können Sie die zu sendenden Daten (vor dem Senden) vom lokalen Code in den Code des Empfängers konvertieren (siehe Abschnitt „Convert_Outgoing - Konvertieren von lokalem Code in den Code des Empfängers“ auf Seite 104).

- Automatische Code-Konvertierung vom Trägersystem UPIC
  Die automatische Code-Konvertierung schalten Sie für die Kopplung zu einem bestimmten Server über die Conversation Characteristic *CHARACTER_CONVERTION* ein. *CHARACTER_CONVERTION* kann wie folgt eingeschaltet werden:

  – indem Sie im Side Information-Eintrag oder in der `upicfile` für diesen Server ein entsprechendes Kennzeichen setzen (siehe Abschnitt „Side Information für standalone UTM-Anwendungen“ auf Seite 298),

  – oder über den Set_Character_Convertion-Aufruf.

  UPIC konvertiert bei eingeschalteter Code-Konvertierung alle Daten, die von diesem Server eintreffen, vor der Übergabe an das Client-Programm in den lokal verwendeten Code, und alle Daten, die vom Client-Programm an den Server gesendet werden, vor dem Senden in den Code des Servers. Das Client-Programm muss sich um die Konvertierung nicht mehr kümmern; *Convert_Incoming* und *Convert_Outgoing* dürfen nicht mehr durchlaufen werden.

  Durch die automatische Code-Konvertierung wird die Möglichkeit geschaffen, mit einem einzigen CPI-C-Programm sowohl mit einer UTM-Anwendung in Unix- oder Windows-Systemen auf Basis des ASCII-Codes als auch mit einer UTM-Anwendung im BS2000/OSD auf Basis des EBCDIC-Codes zu kommunizieren (falls die Benutzerdaten keine Binärinformation enthalten, die bei der Codeumsetzung verfälscht würden).



VORSICHT!

Beachten Sie, dass die Nachrichten bei einer heterogenen Kopplung nur einmal konvertiert werden. Es dürfen nur Nachrichten konvertiert werden, die abdruckbare Zeichen enthalten. Bei einer homogenen Kopplung und bei der Kopplung Windows-System <-> Unix-System darf gar nicht konvertiert werden.

Das Euro-Zeichen hat im Windows-Zeichensatz den Wert 0x80, im ASCII Zeichensatz den Wert 0xa4 und ist im EBCDIC Zeichensatz das allgemeine Währungssymbol mit dem Wert 0x9f. Im strengen Sinne sind dies keine abdruckbaren Zeichen (8-Bit ASCII-Zeichensatz).



Bei UPIC auf Unix-Systemen und UPIC auf BS2000/OSD können Sie zusätzlich die mitgelieferten Konvertierungstabellen nach Ihren eigenen Erfordernissen ändern, compilieren und mit den üblichen Betriebssystem-Mitteln in den UPIC-Bibliotheken austauschen.

4.3.7 Benutzerdefinierte Code-Konvertierung für Windows-Systeme

Die Konvertierungstabellen sind in eine eigene dynamische Bibliothek ausgegliedert. Sie können dadurch die Konvertierungstabellen Ihren eigenen Bedürfnissen anpassen. Im Verzeichnis *upic-dir*\utmcnv32 sind die dazu notwendigen Dateien installiert. Es handelt sich dabei um:

– `kcsaeea.c`, C-Source mit den Konvertierungstabellen der bisherigen UPIC-Versionen. Diese Datei ist aus Kompatibilitätsgründen in der mit dem Produkt ausgelieferten utmcnv32.dll enthalten.

– `kcxaent.c`, C-Source mit vollständigen Konvertierungstabellen zwischen Windows-Zeichensatz und EBCDIC.

– `utmcnv32.def`, Def-Datei mit EXPORT-Anweisungen.

– `utmcnv32.rc`, `resource.h`, Resource-Dateien mit Versionsinformationen.

i Zum Erstellen der Bibliothek, sind die Versionsinformationen nicht zwingend erforderlich.

Mit einem Microsoft Visual C++ Developer Studio gehen Sie folgendermaßen vor:

► Erstellen Sie ein neues Projekt mit dem Namen `utmcnv32` im Verzeichnis *upic-dir*`\utmcnv32`. Das Projekt muss vom Typ Dynamik-Link-Library sein.

► Fügen Sie die Dateien `kcsaeea.c` oder `kcxaent.c`, `utmcnv32.def` und gegebenenfalls `utmcnv32.rc` zu dem Projekt hinzu.

► Erzeugen Sie mit diesem Projekt `utmcnv32.dll`.

► Wenn die Bibliothek `utmcnv32.dll` erfolgreich erstellt wurde, müssen Sie sie noch in das Verzeichnis kopieren, in dem die UPIC-Bibliothek `upicw32.dll` bzw. `upicws32.dll` steht, die von Ihrer Anwendung geladen wird. Das Verzeichnis wird meistens das Windows-Systemdirectory sein (z.B. \Windows\System oder Winnt\System32).

► Vergewissern Sie sich, dass die ursprüngliche Bibliothek `utmcnv32.dll` entweder beim Kopieren überschrieben wird, oder gelöscht wurde, damit sie nicht fälschlicherweise anstelle der neuen Bibliothek vom System geladen wird.

Die Konvertierungstabellen sind in Form zweier Character Arrays der Größe 256 aufgebaut:

– `unsigned char kcsaebc[256]` zur Konvertierung von ASCII Zeichen nach EBCDIC

– `unsigned char kcseasc[256]` zur Konvertierung von EBCDIC Zeichen nach ASCII

Der EBCDIC-Code des ASCII-Zeichens n ist der Wert des n-ten Elements des Characters Array `kcsaebc`, also von `kcsaebc[n]`.

Beispiele

1. „M“ hat den ASCII-Code 4D hexadezimal oder 77 dezimal. `kcsaebc[77]` ist auf den Wert D4 hexadezimal gesetzt, das ist der EBCDIC-Code von „M“.
2. Wenn Sie den Umlaut „Ä“ (Code C4 hexadezimal in ISO 8859-1, Code 63 hexadezimal in EBCDIC.DF.04-1), der in den ursprünglichen Konvertierungstabellen nicht berücksichtigt wird, in Ihre eigenen Konvertierungstabellen einbauen wollen, so müssen Sie den Wert von `kcsaebc[196]` von 0xff nach 0xc4 ändern (ASCII → EBCDIC Konvertierung) und den Wert von `kcseasc[99]` von 0x1a nach 0xc4 ändern (EBCDIC → ASCII Konvertierung).

4.4 Kommunikation mit dem openUTM-Server

In diesem Abschnitt zeigen Beispiele, wie ein CPI-C-Programm mit Einschritt- und Mehrschritt-Vorgängen einer UTM-Anwendung kommunizieren kann. Bei einem Mehrschritt-Vorgang wird in der UTM-Anwendung eventuell mehr als eine Transaktion ausgeführt. Dies kann auch eine verteilte Transaktionsverarbeitung beinhalten (siehe Bild 13 auf Seite 76).

Die in den folgenden Beispielen verwendeten Aufrufe haben folgende Bedeutung:

– Anmelden an das Trägersystem UPIC (*Enable_UTM_UPIC*)

– Initialisieren der Conversation Characteristics (*Initialize_Conversation*)

– Einrichten der Conversation (*Allocate*)

– Senden von Daten (*Send_Data*; Sie können auch *Send_Mapped_Data* verwenden)

– Empfangen der Antwort (*Receive*; Sie können auch *Receive_Mapped_Data* verwenden)

– Abmelden vom Trägersystem UPIC (*Disable_UTM_UPIC*)

Zur Vereinfachung der Darstellung in den Bildern dieses Abschnitts wurde beim Senden und Empfangen die Zwischenspeicherung der Daten im lokalen UPIC-Speicher weggelassen.

4.4.1 Kommunikation mit einem Einschritt-UTM-Vorgang

Die nachfolgenden beiden Bilder zeigen mögliche Formen der Zusammenarbeit zwischen einer CPI-C-Anwendung und einer UTM-Anwendung bei einem Einschritt-Vorgang.

Ein Send- und ein Receive-Aufruf

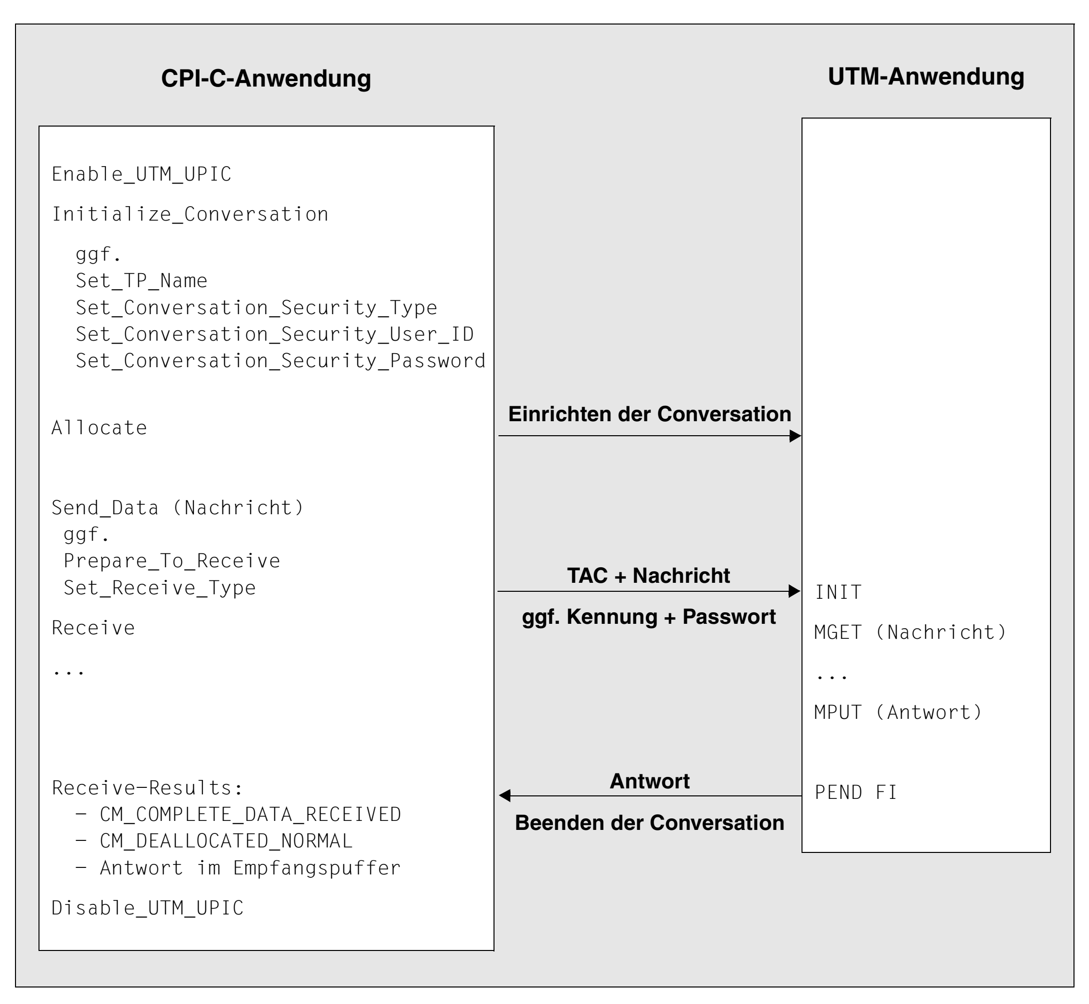


Bild 10: Einschritt-Vorgang mit einem Send-/Receive-Aufruf

Das Programm wartet hier beim *Receive*-Aufruf, bis die Antwort von openUTM eintrifft. Mit CM_COMPLETE_DATA_RECEIVED wird angezeigt, dass die Antwort komplett empfangen wurde. Dass es die letzte und einzige Nachricht war, ist am Returncode CM_DEALLOCATE_NORMAL zu erkennen. Statt *Send_Data* und *Receive* können Sie auch *Send_Mapped_Data* und *Receive_Mapped_Data* verwenden.

Sollen größere Datenmengen übertragen werden, können auch bei Kommunikation mit einem Einschritt-Vorgang mehrere *Send*- und *Receive*-Aufrufe verwendet werden, siehe folgendes Bild.

Mehrere Send- und Receive-Aufrufe

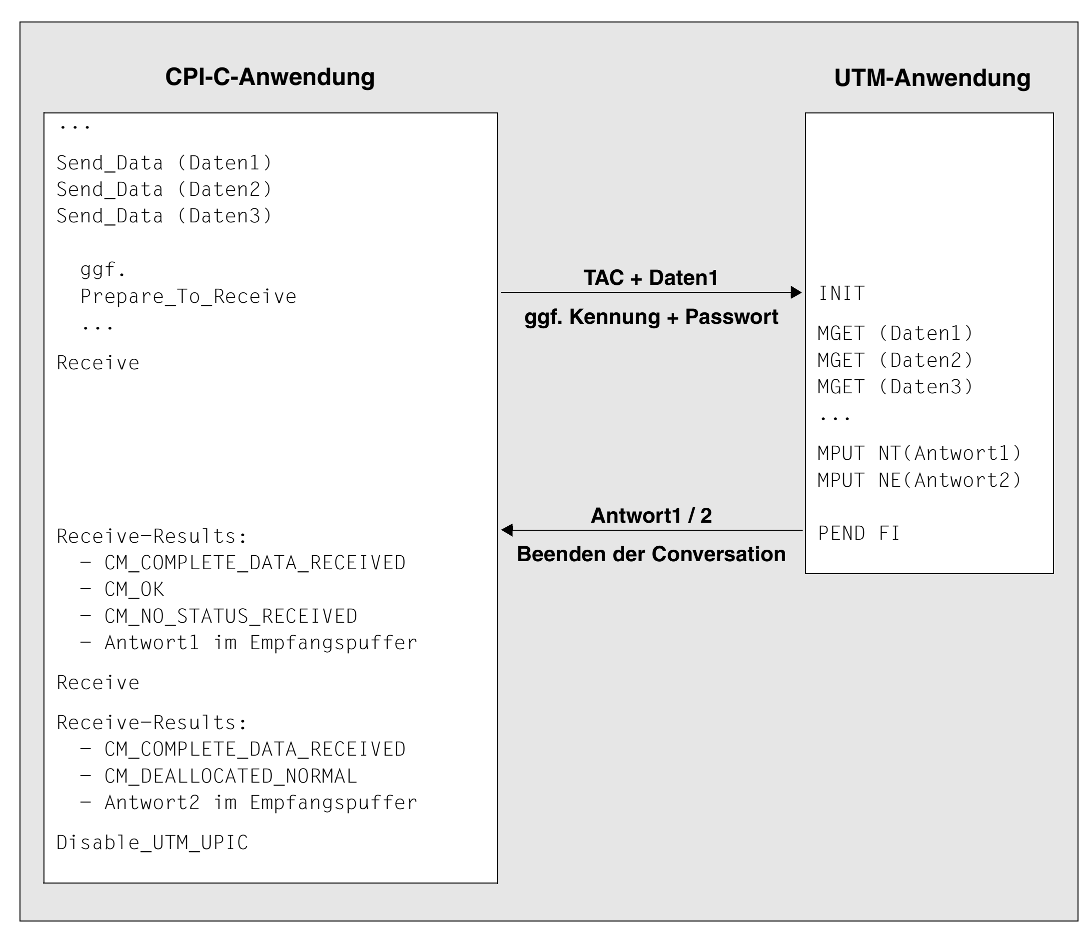


Bild 11: Einschritt-Vorgang mit mehreren Send-/Receive-Aufrufen

Für jeden MPUT-Aufruf wird ein eigener *Receive*-Aufruf durchgeführt.

Nach dem ersten *Receive*-Aufruf wird durch CM_NO_STATUS_RECEIVED zusammen mit CM_OK angezeigt, dass noch weitere Nachrichten vorhanden sind. Deshalb ist ein zweiter *Receive*-Aufruf notwendig, mit dem die zweite und letzte Nachricht empfangen wird. Dass es die letzte Nachricht war, ist an CM_DEALLOCATED_NORMAL zu erkennen.

4.4.2 Kommunikation mit einem Mehrschritt-UTM-Vorgang

Das nachfolgende Bild zeigt eine mögliche Form der Zusammenarbeit zwischen einer CPI-C-Anwendung und einer UTM-Anwendung bei einem Mehrschritt-Vorgang. In diesem Beispiel werden mehrfach Daten gesendet und empfangen.

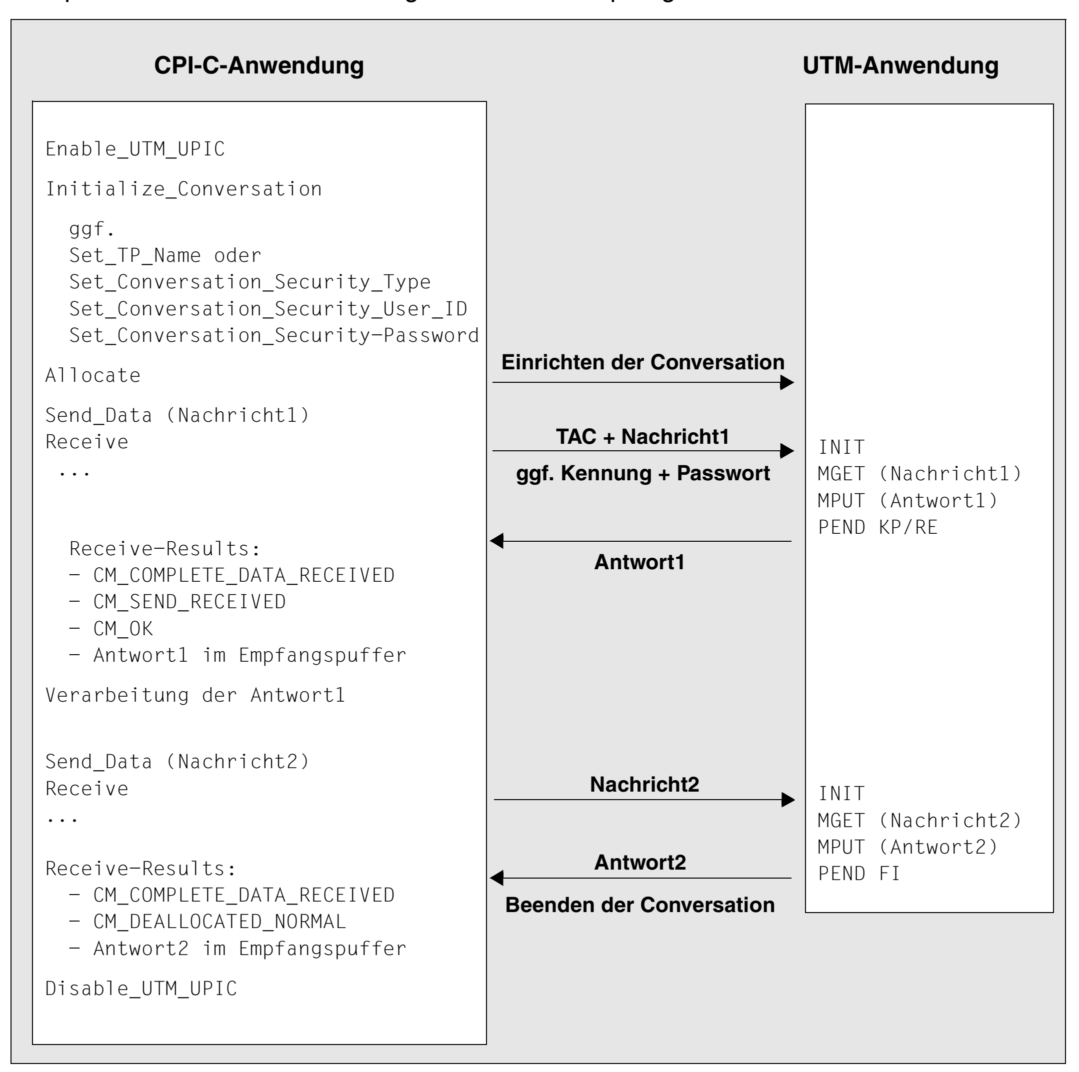


Bild 12: Mehrschritt-Vorgang

Die Kommunikation mit einem Mehrschritt-Vorgang ist dann notwendig, wenn die erste Antwort in der CPI-C-Anwendung verarbeitet werden muss, bevor die zweite Nachricht an openUTM geschickt wird.

4.4.3 Kommunikation mit einem Mehrschritt-UTM-Vorgang unter Nutzung von verteilter Transaktionsverarbeitung

Das nachfolgende Bild zeigt eine mögliche Form der Zusammenarbeit zwischen einer CPI-C-Anwendung und einer UTM-Anwendung bei einem Mehrschritt-Vorgang. In diesem Beispiel wird auf der UTM-Seite eine verteilte Transaktionsverarbeitung zwischen zwei UTM-Anwendungen veranlasst.

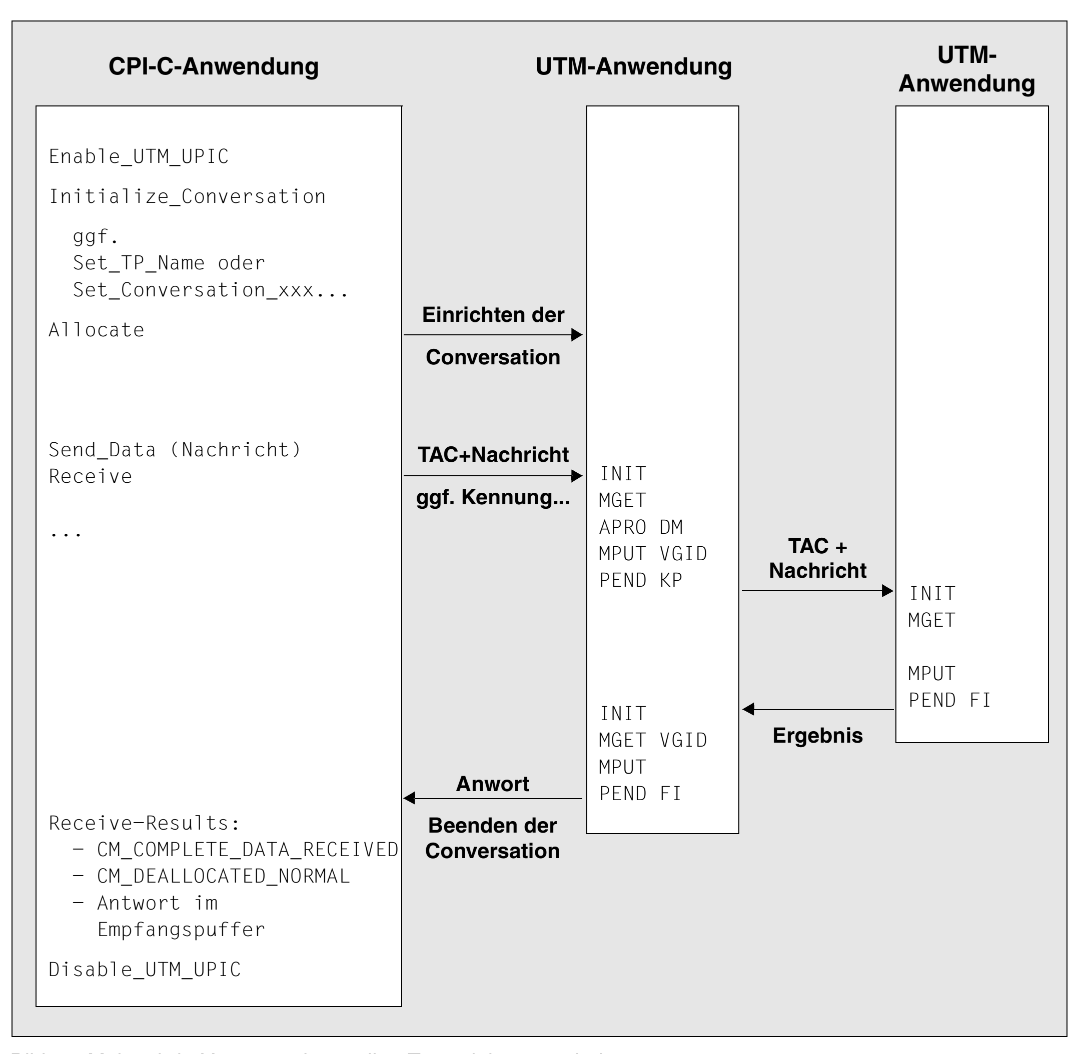


Bild 13: Mehrschritt-Vorgang mit verteilter Transaktionsverarbeitung

4.4.4 Transaktionsstatus abfragen

Mit jeder Benutzernachricht sendet die UTM-Anwendung Informationen über Zustand von Transaktion und Vorgang an den Client. Die CPI-C-Anwendung kann diese Information mit dem Aufruf *Extract_Transaction_State* lesen.

Die Statusinformation wird in einem 4 Bytes langen Feld gesendet. Die ersten beiden Bytes zeigen den Zustand von Vorgang und Transaktion an, die letzten beiden Bytes liefern Diagnoseinformationen, siehe Abschnitt „Extract_Transaction_State - Vorgangs- und Transaktionsstatus des Servers abfragen“ auf Seite 144. Das Programm kann damit z.B. erkennen:

– ob der Verarbeitungsschritt mit oder ohne Transaktionsende abgeschlossen wurde,

– ob zusätzlich der Vorgang beendet wurde

– oder ob die Transaktion zurückgesetzt wurde

Das CPI-C-Programm kann entsprechend darauf reagieren und z.B. den Benutzer ausführlich darüber informieren, ob seine Eingabe erfolgreich übernommen wurde oder ob er sie nochmals an den Server schicken muss, da die Transaktion zurückgesetzt wurde.

4.5 Benutzerkonzept, Security und Wiederanlauf

Beim Trägersystem UPIC kann an der CPI-C- und der XATMI-Schnittstelle das UTM-Benutzerkonzept genutzt werden. Damit stehen bei der Client/Server-Kommunikation die für die Datensicherheit wichtigen Security-Funktionen und Wiederanlauf-Funktionen von openUTM zur Verfügung.

4.5.1 Benutzerkonzept

In einer UTM-Anwendung können UTM-Benutzerkennungen generiert und durch Passwörter einer bestimmten Komplexitätsstufe geschützt werden. Diese Benutzerkennungen sowie Passwörter und deren Komplexitätsstufe müssen in der UTM-Anwendung mit USER-Anweisungen generiert werden. Jede für eine UTM-Anwendung generierte Benutzerkennung kann sowohl von einem Client-Programm als auch von einem Terminalbenutzer verwendet werden.

Das an der CPI-C- und XATMI-Schnittstelle realisierte Benutzerkonzept wirkt für die Dauer einer Conversation, d.h. beim Aufbau jeder Conversation muss das Programm die Berechtigungsdaten (Benutzerkennung und ggf. das Passwort) an openUTM übergeben. In openUTM kann sich ein Client-Programm auch über einen Anmelde-Vorgang (SIGNON-Vorgang; siehe openUTM-Handbuch „Anwendungen programmieren mit KDCS“) anmelden.

Mehrfaches Anmelden unter einer UTM-Benutzerkennung

Ist eine UTM-Benutzerkennung mit Vorgangs-Wiederanlauf (USER ...,RESTART=YES) generiert, dann verknüpft openUTM mit der UTM-Benutzerkennung einen wiederanlauffähigen Vorgangskontext, der über die Benutzerkennung implizit zugeordnet wird.

Unter einer solchen UTM-Benutzerkennung kann zu einer Zeit nur ein Client-Programm oder nur ein Terminalbenutzer mit der UTM-Anwendung arbeiten.

Ist in einer Anwendung, die mehrfaches Anmelden unter einer Benutzerkennung (SIGNON ..., MULTI-SIGNON=YES) erlaubt, eine UTM-Benutzerkennung ohne Wiederanlauf (USER ...,RESTART=NO) generiert, dann ist ein mehrfaches Anmelden unter dieser Benutzerkennung möglich. Hier wird der wiederanlauffähige Vorgangskontext nicht benötigt.

4.5.2 Security-Funktionen

Folgende Security-Funktionen sind in openUTM realisiert:

- Zugangsschutzfunktionen

  Diese Funktionen werden in openUTM durch UTM-Benutzerkennungen und Passwörter einer bestimmten Komplexitätsstufe realisiert. Die Nutzung dieser Funktionen wird bei CPI-C und XATMI wie folgt realisiert:

  – Bei CPI-C gibt es die Aufrufe

    Set_Conversation_Security_Type: Typ des Zugangsschutzes festlegen
    Set_Conversation_Security_User_ID: UTM-Benutzerkennung angeben
    Set_Conversation_Security_Password: Zugehöriges Passwort angeben

  – zusätzlich bei UPIC

    Set_Conversation_Security_New_Password: neues Passwort vergeben

    Diese Aufrufe müssen Sie vor dem Einrichten der Conversation absetzen.

    Falls die Anmeldung nicht erfolgreich war, steht nach einem *Receive*- oder *Receive_Mapped_Data* zusätzlich noch folgender Aufruf zur Verfügung:

    Extract_Secondary_Return_Code: erweiterten Returncode abfragen

  – An der Schnittstelle XATMI gibt es beim Aufruf *tpinit()* entsprechende Parameter, mit denen diese Zugangsschutzfunktionen aktiviert werden (siehe Abschnitt „tpinit - Client initialisieren“ auf Seite 261).

  Sobald das CPI-C- oder XATMI-Programm diese Aufrufe verwendet, werden implizit auch die nachfolgend geschilderten Zugriffsschutz- und Datensicherheitsfunktionen wirksam.

- Zugriffsschutzfunktionen

  Damit bestimmte Services der UTM-Server-Anwendung nur einem ausgewählten Benutzerkreis zugänglich sind, können Sie wahlweise das Lock-/Keycode-Konzept oder das Access-List-Konzept von openUTM verwenden (siehe openUTM-Handbuch „Konzepte und Funktionen“):

  – Beim Lock/Keycode-Konzept können den Transaktionscodes (Services) und den LTERM-Partnern der UTM-Server-Anwendung Lockcodes zugeordnet werden. Nur Benutzer oder Clients, deren Benutzerkennungen die entsprechenden Keycodes zugeordnet sind, können auf diese Objekte zugreifen. Bei der Generierung wird der Benutzerkennung ein Keyset mit einem oder mehreren Keycodes zugeordnet (USER ...,KSET=Keyset-Name). Das Keyset legt fest, auf welche Services der UTM-Anwendung der Client zugreifen darf.

  – Beim Access-List-Konzept werden Rollen in Form von Keycodes definiert. Die Transaktionscodes werden mit Access-Lists geschützt. Jeder Benutzerkennung werden eine oder mehrere Rollen zugeordnet (Generierunganweisung USER ...,KSET=). Ein Client darf über eine bestimmte Benutzerkennung nur dann auf einen Service zugreifen, wenn mindestens eine seiner Rolle in der Access-List enthalten ist. Zusätzlich können auch LTERM-Partnern Rollen zugeordnet werden, dann gilt Entsprechendes für den Zugriff über einen LTERM-Partner.

- Datensicherheit durch Benutzer-spezifische Langzeitspeicher (ULS)

  Per Generierung kann jeder UTM-Benutzerkennung ein Benutzer-spezifischer Langzeitspeicher zugeordnet werden. Auf diesen Speicher können Teilprogramme des Benutzers/Clients und vom Administrator gestartete Programme zugreifen, wobei konkurrierende Zugriffe von openUTM synchronisiert werden. Die Informationen im ULS bleiben über das Vorgangsende hinaus erhalten. Sie werden nicht gelöscht, sondern können nur durch Null-Nachrichten überschrieben werden. Der ULS dient zur Übergabe von Daten zwischen Vorgängen und Programmen des Benutzers.

  Mit der KDCDEF-Steueranweisung ULS wird jeder Benutzerkennung der UTM-Anwendung ein Benutzer-spezifischer Langzeitspeicher ULS zugeordnet.

Innerhalb von openUTM werden Security-Funktionen in einer Client/Server-Umgebung wie folgt realisiert:

1. Vor dem Start eines UTM-Services werden die Berechtigungsdaten, die vom Client kommen, validiert und es wird die entsprechende UTM-Benutzerkennung zusammen mit dem dazugehörigen Keyset zugeordnet. Dies entspricht etwa einem KDCSIGN eines Terminalbenutzers unmittelbar vor dem Vorgangsstart.

   Falls die Gültigkeitsdauer des Benutzerpassworts abgelaufen ist und die UTM-Anwendung mit Grace-Sign-On generiert ist, dann ist eine Anmeldung immer noch möglich, siehe Seite 80.

2. Wird das Lock/Keycode oder das Access-List-Konzept eingesetzt, dann prüft openUTM, ob der Service unter dieser Benutzerkennung und über diesen LTERM-Partner gestartet werden darf. Wenn ja, dann erscheint im UTM-Service die vom Client übergebene UTM-Benutzerkennung im Kopf des Kommunikationsbereichs (KB-Kopf). Die mit dieser UTM-Benutzerkennung verknüpften Berechtigungen (Keyset) sind wirksam.
3. Die ULS-Blöcke, die der vom Client übergebenen UTM-Benutzerkennung zugeordnet sind, können verwendet werden. Melden sich mehrere Clients unter einer Benutzerkennung an, dann verwenden diese gemeinsam denselben ULS-Block, da pro Benutzerkennung immer nur ein ULS-Block existiert.
4. Am Ende des Vorgangs wird die Zuordnung (1. bis 3.) wieder aufgehoben.

Anmeldung nach Ablauf des Passwortes (Grace-Sign-On)

Ist die UTM-Anwendung mit Grace-Sign-On generiert, dann kann sich ein Client auch nach Ablauf des Passwortes noch an die Anwendung anmelden. Ist für den UPIC-Client kein Anmelde-Vorgang generiert, so erhält das Programm in diesem Fall nach einem *Receive*- oder *Receive_Mapped_Data*-Aufruf den Returncode CM_SECURITY_NOT_VALID. Zusätzliche Informationen werden in Form eines sekundären Returncodes geliefert. Dieser enthält bei abgelaufenem Passwort einen der folgenden Werte:

– CM_SECURITY_PWD_EXPIRED_RETRY, wenn die Anwendung mit Grace-Sign-On generiert ist. In diesem Fall kann das Programm beim nächsten Anmelden mit *Set_Conversation_Security_New_Password* ein neues Passwort setzen. Dies muss sich vom bisherigen Passwort unterscheiden und den gleichen Anforderungen wie das bisherige genügen (Länge, Komplexität wie z.B. Sonderzeichen).

– CM_SECURITY_PWD_EXPIRED_NO_RETRY, wenn die Anwendung nicht mit Grace-Sign-On generiert ist. In diesem Fall kann der Client-Benutzer sich nicht mehr unter diese UTM-Benutzerkennung anmelden. Er muss den Administrator der UTM-Anwendung darum bitten, ein neues Passwort für ihn einzutragen.

Der sekundäre Returncode eines *Receive*- oder *Receive_Mapped_Data*-Aufrufs kann auch mit einem nachfolgenden CPI-C-Aufruf *Extract_Secondary_Returncode* abgefragt werden. *Extract_Secondary_Returncode* gibt den sekundären Returncode des letzten *Receive*- oder *Receive_Mapped_Data*-Aufrufs zurück.

4.5.3 Wiederanlauf

Ein echter Wiederanlauf ist nur mit der CPI-C-Schnittstelle von UPIC möglich, da nur diese mit Mehrschritt-UTM-Vorgängen kommunizieren kann. Bei der Schnittstelle XATMI kann allerdings ebenfalls die letzte Ausgabenachricht gelesen werden, siehe Abschnitt „Wiederanlauf“ auf Seite 252. Die folgende Beschreibung bezieht sich daher nur auf CPI-C-Client-Programme.

Mit der UTM-Benutzerkennung ist ein Vorgangskontext verbunden. Der Vorgangskontext enthält z.B. die letzte Ausgabenachricht und Vorgangsdaten wie KB und LSSBs usw. Zusätzlich kann der Client auch einen Client-Kontext an die UTM-Anwendung senden,siehe Abschnitt „Wiederanlauf mit Client-Kontext“ auf Seite 83.

Die Wiederanlauffähigkeit hängt davon ab, wie eine UTM-Benutzerkennung generiert ist:

– Ist eine UTM-Benutzerkennung mit USER ...,RESTART=YES (Standardwert) generiert, dann führt openUTM nach Systemausfällen oder nach dem Verlust der Verbindung zum Client einen Vorgangs-Wiederanlauf durch. D.h. openUTM reaktiviert für die Benutzerkennung den Vorgangskontext und ggf. den Client-Kontext.

– Ist eine UTM-Benutzerkennung mit RESTART=NO generiert, dann führt openUTM keinen Vorgangs-Wiederanlauf durch. Auch nicht, wenn der vom Client verwendete LTERM-Partner mit LTERM ...,RESTART=YES generiert ist.

Vorgangs-Wiederanlauf heißt: Nach erneutem Anmelden des Clients setzt die Verarbeitung am letzten Sicherungspunkt eines noch offenen Vorgangs wieder auf. openUTM überträgt die letzte Nachricht des noch offenen Vorgangs und ggf. den Client-Kontext erneut an den Client. Dieser kann den Vorgang dann weiterführen.

Existiert unter der Benutzerkennung ein offener Vorgang für den Client, dann muss dieser Vorgang unmittelbar nach dem nächsten Anmelden weitergeführt werden, sonst beendet openUTM den offenen Vorgang abnormal.

Das Client-Programm muss den Wiederanlauf initiieren, indem es zuerst eine neue Conversation aufbaut und dabei beim Aufruf *Set_TP_Name* den Transaktionscode KDCDISP übergibt. Das folgende Beispiel skizziert ein solches „Wiederanlauf-Programm“ für CPI-C.

Beispiel

```
Initialize_Conversation (...)
Set_Conversation_Security_Type (...,CM_SECURITY_PROGRAM,..)           1.
Set_Conversation_Security_User_ID (...,"UTMUSER1",..)                 1.
Set_Conversation_Security-Password (...,"SECRET",..)                  1.
Set_TP_Name (...,"KDCDISP",...)                                       2.

Allocate (...)

Send_Data (...)                                                       3.
          /* Leere Nachricht */

Receive (...)

  return_code=CM_OK
          /* Vorgang offen, Senderecht geht an Client */
          /* Kommunikation mit UTM-Vorgang fortsetzen */
  status_received=CM_SEND_RECEIVED                                    4.
```

oder

```
  return_code=CM_DEALLOCATED_NORMAL                                   5.
          /* Vorgangsende, Wiederanlauf beendet       */
```

oder

```
  return_code=CM_TP_NOT_AVAILABLE_NO_RETRY                            6.
          /* Wiederanlauf nicht möglich               */
```

1. Das Programm benutzt die Zugangsschutz-Funktionen von openUTM und setzt explizit die UTM-Benutzerkennung und das Passwort.
2. Das Programm muss für den Wiederanlauf *TP_name* auf KDCDISP setzen.
3. Bei *Send_Data* dürfen keine Daten gesendet werden, d.h. *send_length* muss auf 0 gesetzt sein („Leere Nachricht“).
4. Die Verarbeitung und die Kommunikation mit dem UTM-Vorgang können fortgesetzt werden.
5. Das Programm hat bereits die letzte Ausgabenachricht erhalten, auf UTM-Seite ist kein Vorgang mehr offen.
6. Wegen UTM-Neugenerierung ist kein Wiederanlauf möglich.

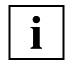
Als Ergebnis eines solchen Wiederanlauf-Programms erhält der Client beim *Receive* immer die letzte Ausgabenachricht von openUTM.

Ein Benutzer kann sich unter einer bestimmten Benutzerkennung auf verschiedene Arten an einem UTM-Server anmelden:

– von einem Terminal aus
– über einen Transportsystem-Client
– über Client-Programme mit verschiedenen Trägersystemen

Ein Wiederanlauf durch ein Client-Programm ist nur möglich, wenn die Benutzerkennung zuletzt auch von einem Client-Programm mit demselben Trägersystem verwendet wurde. Ist dies nicht der Fall, dann lehnt openUTM die Anmeldung des Client-Programms ab (CM_SECURITY_NOT_VALID), da der offene Vorgang zunächst von dem Partner beendet werden muss, der ihn gestartet hat.

Ist beim Conversation-Aufbau mit KDCDISP kein offener Vorgang vorhanden, so beendet openUTM die Conversation nach dem Senden der letzten Ausgabenachricht des vorherigen Vorgangs. Wurde der letzte Vorgang von einem anderen Partner gestartet, dann übergibt openUTM keine Nachricht (Returncode CM_TP_NOT_AVAILABLE_NO_RETRY).



Um die genannten Probleme zu vermeiden, sollte eine mit RESTART=YES generierte UTM-Benutzerkennung entweder nur von Client-Programmen mit gleichem Trägersystem oder nur von Terminalbenutzern verwendet werden.

Ist nach einer Neugenerierung der UTM-Anwendung kein Anwendungskontext vorhanden, dann erhält das Programm den Returncode CM_TP_NOT_AVAILABLE_NO_RETRY. openUTM beendet die Conversation.

Offene Vorgänge eines Client mit Wiederanlauffähigkeit werden vom openUTM-Tool KDCUPD ab der openUTM Version 5.1 übertragen.

Wiederanlauf mit Client-Kontext

Der Client kann mit jeder Benutzernachricht einen so genannten „Client-Kontext“ an die UTM-Anwendung schicken. Ein Client-Kontext besteht aus einer maximal 8 Byte langen Zeichenkette. Dies kann z.B. die Uhrzeit oder eine Nachrichten-ID sein.

Falls die Benutzerkennung mit RESTART=YES generiert ist, dann wird der Client-Kontext von openUTM so lange gesichert, bis die Conversation beendet oder bis der Kontext durch einen neuen Kontext überschrieben wurde.

Wenn der Client einen Wiederanlauf anfordert, dann überträgt openUTM den Client-Kontext zusammen mit der letzten Dialog-Nachricht an den Client. Damit kann das Programm anhand des Client-Kontexts eindeutig feststellen, an welcher Stelle im Dialog der Wiederanlauf stattfindet und wie es darauf reagieren muss, z.B. durch Ausgabe eines bestimmten Formulars. Zum Setzen und Lesen des Client-Kontexts gibt es folgende UPIC-Aufrufe:

Set_Client_Context: Client-Kontext setzen

Extract_Client_Context: Den letzten von openUTM gesendeten Client-Kontext ausgeben

4.6 Verschlüsselung

Clients greifen häufig über offene Netze auf UTM-Services zu. Damit besteht die Möglichkeit, dass Unbefugte auf der Leitung mitlesen und z.B. Passwörter für UTM-Benutzerkennungen oder sensible Benutzerdaten ermitteln. Um dies zu verhindern, unterstützt openUTM die Verschlüsselung von Passwörtern und Benutzerdaten für Client-Verbindungen.

Die Verschlüsselung kann bei openUTM auch dazu verwendet werden, den Zugang durch Clients und den Zugriff auf bestimmte Services zu kontrollieren. openUTM verwendet zum Verschlüsseln eine Kombination aus symmetrischem AES- oder DES-Schlüssel und asymmetrischem RSA-Schlüssel.

Verschlüsselungsverfahren

Passwörter und Benutzerdaten auf einer Verbindung werden mit einem symmetrischen Schlüssel verschlüsselt. Dies ist entweder ein AES- oder ein DES-Schlüssel. Client und UTM-Anwendung verwenden denselben symmetrischen Schlüssel zum Ver- und Entschlüsseln der Nachrichten. Dieser Schlüssel wird vom Client erzeugt und beim Verbindungsaufbau an die UTM-Anwendung übertragen, der Schlüssel wird nur für diese eine Verbindung verwendet.

Zur Erhöhung der Sicherheit wird der AES- bzw. DES-Schlüssel selbst verschlüsselt übertragen. Dazu werden für die UTM-Anwendung bei der Generierung ein oder mehrere RSA-Schlüsselpaare erzeugt. Ein RSA-Schlüsselpaar besteht aus einem öffentlichen und einem privaten Schlüssel. Der öffentliche Schlüssel wird direkt beim Aufbau der Verbindung von der UTM-Anwendung an den Client übertragen. Der Client verschlüsselt damit den AES- bzw. DES-Schlüssel. Zur Entschlüsselung dieses Schlüssels verwendet die UTM-Anwendung den privaten Schlüssel, der nur der UTM-Anwendung bekannt ist.

In openUTM werden abhängig von der Generierung bis zu vier verschiedene RSA-Schlüsselpaare der Modulo-Länge 200, 512, 1024 und 2048 erzeugt.

Mit Hilfe dieser Schlüssel lassen sich über die UTM-Generierung (Operand ENCRYPTION-LEVEL) unterschiedliche Verschlüsselungsebenen festlegen, siehe Tabelle.

Generierte Verschlüsselungsebene	**Modulo-Länge des RSA-Schlüssels**	**Symmetrischer Schlüssel**
TRUSTED	kein Schlüssel	kein Schlüssel
NONE	situationsabhängig	situationsabhängig
1	200	DES
2	512	AES
3	1024	AES
4	2048	AES

Tabelle 7: Generierte Verschlüsselungsebenen und zugehörige Schlüssel

Jedes RSA-Schlüsselpaar kann in openUTM per Administration geändert und aktiviert werden. Nur aktivierte RSA-Schlüssel werden auch verwendet. Zusätzlich besteht für den UPIC-Client die Möglichkeit, den öffentlichen Schlüssel vorab lokal zu hinterlegen. Beim Verbindungsaufbau wird der empfangene öffentliche Schlüssel anhand des hinterlegten öffentlichen Schlüssels überprüft.

Der aktive RSA-Schlüssel kann über Aufrufe der UTM-Administrationsschnittstelle oder mit dem Administrationstool openUTM-WinAdmin ausgelesen und gelöscht werden.

Voraussetzungen

Voraussetzung für die Verschlüsselung zwischen openUTM und UPIC-Clients ist, dass auf beiden Seiten die Lizenz zum Verschlüsseln vorhanden ist. Aus rechtlichen Gründen werden die Verschlüsselungs-Funktionen von openUTM als eigenes Produkt (openUTM-Crypt) ausgeliefert, das separat installiert werden muss.

Wenn bei openUTM für diesen Partner eine Verschlüsselungsebene 1 bis 4 generiert ist, diese Voraussetzungen jedoch nicht erfüllt sind, dann wird der Verbindungsaufbau abgelehnt. Dies kann im Einzelnen folgende Gründe haben:

- Der Client unterstützt keine Verschlüsselung, weil die Verschlüsselungsfunktionalität nicht installiert ist oder weil die UPIC-Version eine Exportversion ist.
- openUTM selbst kann mangels geeigneter Verschlüsselungsbibliothek (Exportversion) nicht verschlüsseln.

Ablauf

Beim Verbindungsaufbau des Client zur UTM-Anwendung erhält openUTM vom Client die Information, ob er Verschlüsselung unterstützt.

Wenn die Verbindung zwischen Client und Server zustande gekommen ist und von beiden Partnern Verschlüsselung unterstützt wird, dann sendet der Client an den Server die Information, bis zu welcher Verschlüsselungsebene er die Verschlüsselung unterstützt. Der Server vergleicht dies mit seiner Generierung für diesen Partner.

Abhängig von der Verschlüsselungsebene, die für den Client in der UTM-Anwendung generiert ist, können unterschiedliche Situationen auftreten:

ENCRYPTION-LEVEL=TRUSTED
: Der Client ist als vertrauenswürdig generiert. In diesem Fall fordert openUTM keine Verschlüsselung an. Der Client kann auch keine Verschlüsselung erzwingen.

ENCRYPTION-LEVEL=NONE
: In diesem Fall sendet die UTM-Anwendung den RSA-Schlüssel mit der größten Modulo-Länge an den Client. Dieser RSA-Schlüssel bestimmt die Verschlüsselungsebene.

  Je nach Länge des erhaltenen RSA-Schlüssels erzeugt der Client einen AES-Schlüssel (bei RSA-Schlüssellänge ≥ 512) oder einen DES-Schlüssel (bei RSA-Schlüssellänge = 200). Der Client verschlüsselt den AES- bzw. DES-Schlüssel mit dem RSA-Schlüssel und schickt ihn an den Server zurück. openUTM speichert den Schlüssel für die spätere Verwendung auf dieser Verbindung.

  Es werden standardmäßig nur Passwörter verschlüsselt.

  Der Client kann jedoch die Verschlüsselung der Benutzerdaten über das Schlüsselwort ENCRYPTION_LEVEL in der `upicfile` oder über den Aufruf *Set_Conversation_Encryption_Level* erzwingen.

  Hinweis

  Wenn die Verschlüsselungsfunktionalität nicht installiert ist, dann werden Passwörter und Benutzerdaten unverschlüsselt ausgetauscht.

ENCRYPTION-LEVEL=1

Der öffentliche RSA-Schlüssel der Modulo-Länge 200 wird an den Client gesendet. Der Client erzeugt einen DES-Schlüssel, verschlüsselt ihn mit dem RSA-Schlüssel und und sendet ihn zurück. openUTM speichert den DES-Schlüssel für die spätere Verwendung.

Es werden Passwörter und Benutzerdaten verschlüsselt.

Der Aufruf *Set_Conversation_Encryption_Level* oder der Eintrag ENCRYPTION_LEVEL in der `upicfile` haben keine Wirkung.

ENCRYPTION-LEVEL=2, 3 oder 4

Der UTM-Server sendet den öffentlichen RSA-Schlüssel, der zu der jeweiligen Ebene gehört. Dieser hat die Länge 512, 1024 oder 2048, siehe Tabelle 7 auf Seite 85.

Der Client erzeugt einen AES-Schlüssel, verschlüsselt ihn mit dem RSA-Schlüssel und sendet ihn an den Server zurück. openUTM speichert den AES-Schlüssel für die spätere Verwendung auf dieser Verbindung.

Es werden Passwörter und Benutzerdaten verschlüsselt.

Der Aufruf *Set_Conversation_Encryption_Level* oder der Eintrag ENCRYPTION_LEVEL in der `upicfile` haben keine Wirkung.

Die Verschlüsselungsebene *client-level* der Conversation kann mit dem Aufruf *Extract_Conversation_Encryption_Level* ausgelesen werden, am besten nach dem Aufruf *Allocate*.

Verschlüsselung bei geschütztem TAC

Ein Vorgang einer UTM-Anwendung kann geschützt werden, indem dem zugehörigen TAC per Generierung im Operanden ENCRYPTION-LEVEL=*tac-level* eine Verschlüsselungsebene zugeordnet wird. D.h., dass ein Client den so geschützten Vorgang nur dann aufrufen kann, wenn die Daten entsprechend verschlüsselt übertragen werden. Abhängig von der Generierung des Client und der Verschlüsselungsebene des TAC können dabei folgende Situationen auftreten:

TRUSTED ist für den Client generiert

openUTM fordert keine Verschlüsselung an, der Client kann auch geschützte Vorgänge starten. Der Client kann keine Verschlüsselung erzwingen, da keine Schlüssel ausgetauscht wurden.

NONE ist für den Client generiert

openUTM fordert in diesem Fall keine Verschlüsselung an.

Wenn sich beim Verbindungsaufbau eine Verschlüsselungsebene *client-level* > 0 ergeben hat und wenn eine Conversation initialisiert wird, deren TAC Verschlüsselung der Ebene 1 oder 2 erfordert (*tac-level*), dann gibt es folgende Möglichkeiten:

– *client-level ≥ tac-level*
  wobei der Client für diese Conversation die Verschlüsselung eingeschaltet hat.

  Der Vorgang kann gestartet werden. Der Client sendet schon die ersten Benutzerdaten verschlüsselt.

– *client-level ≥ tac-level*
  wobei der Client für diese Conversation von sich aus **keine** Verschlüsselung eingeschaltet und noch keine Benutzerdaten gesendet hat.

  Der Vorgang kann gestartet werden. Die UTM-Anwendung übergibt alle Ausgaben verschlüsselt auf der Ebene *client-level* an den Client. Der Client verschlüsselt nun ebenfalls alle weiteren Nachrichten an openUTM auf der Ebene *client-level*.

– *client-level* < *tac-level*
  Der UPIC-Client hat schon Benutzerdaten gesendet, die entweder nicht verschlüsselt waren oder mit der niedrigeren Verschlüsselungsebene verschlüsselt waren.

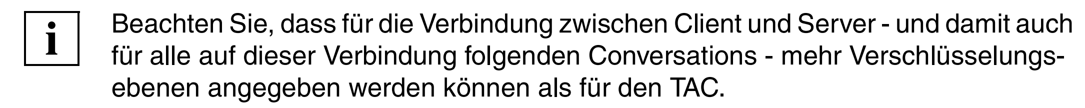
  openUTM beendet die Conversation.

1, 2, 3 oder 4 ist für den Client generiert

Wird eine Conversation initialisiert, deren TAC Verschlüsselung der Ebene 1 oder 2 erfordert (*tac-level*), dann gibt es folgende Möglichkeiten:

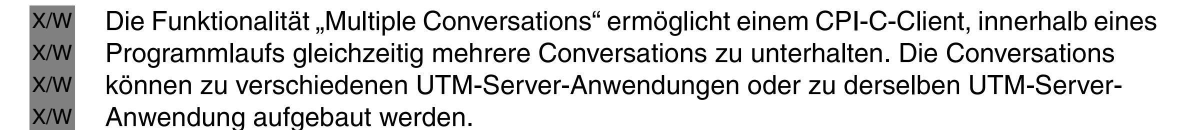
– *client-level ≥ tac-level*
  Der Vorgang kann gestartet werden.

– *client-level* < *tac-level*
  Der Vorgang kann nicht gestartet werden, openUTM beendet die Conversation.

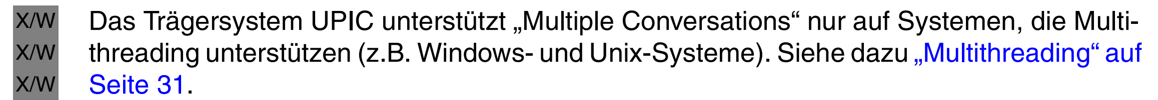


Beachten Sie, dass für die Verbindung zwischen Client und Server - und damit auch für alle auf dieser Verbindung folgenden Conversations - mehr Verschlüsselungsebenen angegeben werden können als für den TAC.

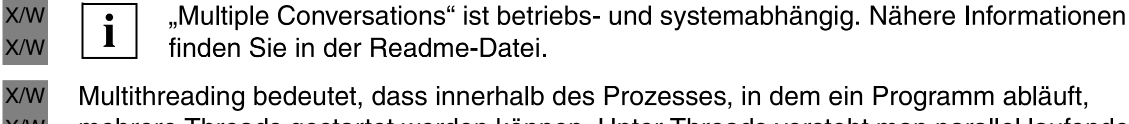
4.7 Multiple Conversations



Die Funktionalität „Multiple Conversations“ ermöglicht einem CPI-C-Client, innerhalb eines Programmlaufs gleichzeitig mehrere Conversations zu unterhalten. Die Conversations können zu verschiedenen UTM-Server-Anwendungen oder zu derselben UTM-Server-Anwendung aufgebaut werden.



Das Trägersystem UPIC unterstützt „Multiple Conversations“ nur auf Systemen, die Multithreading unterstützen (z.B. Windows- und Unix-Systeme). Siehe dazu „Multithreading“ auf Seite 31.



i „Multiple Conversations“ ist betriebs- und systemabhängig. Nähere Informationen finden Sie in der Readme-Datei.

Multithreading bedeutet, dass innerhalb des Prozesses, in dem ein Programm abläuft, mehrere Threads gestartet werden können. Unter Threads versteht man parallel laufende Programmteile innerhalb eines Prozesses, in denen Verarbeitungsschritte unabhängig voneinander bearbeitet werden können. Threads werden deshalb auch oft nebenläufige Prozesse genannt. Der Einsatz von Threads entspricht quasi einem Multi-Processing, das vom Programm selbst verwaltet wird und im selben Prozess abläuft wie das Programm selbst.

CPI-C-Clients, die auf Systemen mit Multithreading ablaufen und entsprechend implementiert sind, können also zu einem Zeitpunkt mit mehreren UTM-Services gekoppelt werden.

CPI-C-Clients, die auf Systemen ablaufen, die kein Multithreading unterstützen, können zu einem Zeitpunkt nur eine Conversation unterhalten. Erst wenn diese Conversation abgebaut ist, kann eine neue aufgebaut werden.

Wenn eine Client-Anwendung gleichzeitig mehrere Conversations bearbeiten will, dann muss jede einzelne dieser Conversations in einem eigenen Thread unabhängig von den anderen Conversations bearbeitet werden. Dabei müssen Sie folgendes beachten:

– Der erste Thread des Prozesses, in dem die anderen Threads gestartet werden, ist der Main-Thread. Im Main-Thread kann - wie in jedem anderen Prozess - auch eine Conversation aufgebaut werden.

– Für jede weitere Conversation, die das Programm aufbauen und parallel bearbeiten soll, muss explizit ein Thread gestartet werden. Zum Starten der Threads stehen Systemaufrufe zur Verfügung. Diese Systemaufrufe sind abhängig vom Betriebssystem und vom verwendeten Compiler (siehe Beispiel auf Seite 92).

– In jedem der gestarteten Threads muss die Ablaufumgebung für den CPI-C-Client gestartet werden. Dazu muss in jedem Thread ein *Enable_UTM_UPIC*-Aufruf abgesetzt werden. Dabei kann sich das CPI-C-Programm in allen Threads mit demselben oder auch mit verschiedenen Namen anmelden.

X/W – In jedem einzelnen Thread müssen die Conversation Characteristics mit einem *Initialize_Conversation*-Aufruf gesetzt werden. Dabei wird der Conversation von UPIC eine eigene Conversation-ID zugeordnet.

X/W – Jede Conversation-ID kann nur innerhalb des Threads benutzt werden, in dem die zugehörige Conversation initialisiert und aufgebaut wurde. Wird die Conversation-ID in einem anderen Thread bei einem CPI-C-Aufruf angegeben, dann liefert UPIC den Returncode CM_PROGRAM_PARAMETER_CHECK zurück.

X/W – In jedem Thread muss sich das Programm mit *Disable_UTM_UPIC* bei UPIC abmelden, bevor der Thread beendet wird.

X/W – Der Main-Thread darf sich erst beenden, wenn alle anderen Threads abgemeldet und beendet sind.

X/W Die Abläufe innerhalb des Client-Programms sind im folgenden Bild dargestellt.

X/W **i** Upic-Local:
Upic-Local unterstützt die Funktion „Multiple Conversations“ nicht.



Main-Thread

Starten des Prozesses und des Main-Threads:

```
main()
...
```

Starten zweier weiterer Threads in dem Prozess:

```
CreateThread()
CreateThread()
```

Die 3 Threads laufen parallel und unabhängig voneinander ab.

Anmelden des Main-Thread:

```
Enable_UTM_UPIC()
...
```

Aufbau einer Conversation im Main-Thread:

```
Initialize_Conversation()
```

Es wird die Conversation-ID CID0 zurückgeliefert, sie darf nur im Main-Thread verwendet werden.

```
...
Set_...(CID0,....)
Allocate (CID0,...)
Send_Data(CID0,...)
...
```

ggf:

```
Prepare_To_Receive(CID0..)
...
Receive(CID0,...)
...
```

Main-Thread abmelden:

```
Disable_UTM_UPIC()
```

Warten auf Beendigung der anderen Threads

```
WaitForMultiple_Objects()
```

Main-Thread und Prozess beenden:

```
ExitProcess(0)
```

Thread1

Anmelden Thread1:

```
Enable_UTM_UPIC()
...
```

Aufbau einer Conversation im Thread1:

```
Initialize_Conversation()
```

Es wird die Conversation-ID CID1 zurückgeliefert, sie darf nur im Thread1 verwendet werden.

```
...
Set_...(CID1,....)
Allocate (CID1,...)
Send_Data(CID1,...)
...
```

ggf:

```
Prepare_To_Receive(CID1..)
...
Receive(CID1,...)
...
```

Thread1 abmelden:

```
Disable_UTM_UPIC()
```

Thread1 beenden:

```
ExitThread(0)
```

Thread2

Anmelden Thread2:

```
Enable_UTM_UPIC()
...
```

Aufbau einer Conversation im Thread2:

```
Initialize_Conversation()
```

Es wird die Conversation-ID CID2 zurückgeliefert, sie darf nur im Thread2 verwendet werden.

```
...
Set_...(CID2,....)
Allocate (CID2,...)
Send_Data(CID2,...)
...
```

ggf:

```
Prepare_To_Receive(CID2..)
...
Receive(CID2,...)
...
```

Thread2 abmelden:

```
Disable_UTM_UPIC()
```

Thread2 beenden:

```
ExitThread(0)
```



Bild 14: Starten mehrerer Threads innerhalb eines Prozesses
(die grau unterlegte Fläche entspricht dem Prozess, in dem das Client-Programm abläuft)

W **Beispiel für „Multiple Conversation“ (in Visual C++ für Windows-Systeme)**

W Das zu diesem Schema gehörende Client-Programm hat den folgenden Aufbau:

```
void main ()                                                       1.
{
   ...
   thrd[0] = CreateThread(...,UpicThread,...);                     2.
   thrd[1] = CreateThread(...,UpicThread,...);
   ...
   Enable_UTM_UPIC (...);                                          3.
   ...
   /* Aufrufe zum Aufbauen und Bearbeiten einer Conversation im    */
   /* Main-Thread:                                                 */
   Initialize_Conversation (...)
   ...
   Allocate (...)
   ....
   Send_Data (...)
   ...
   Receive (...)
   ...
   Disable_UTM_UPIC (...);
   ...
   WaitforMultipleObjects(2,&thrd[0],...);                         4.
   ExitProcess (0);                                                5.
}
DWORD WINAPI UpicThread(LPVOID arg)                                6.
{
   ...
   Enable_UTM_UPIC (...);
   ...
   /* Aufrufe zum Aufbauen und Bearbeiten der Conversation im Thread*/
   /* wie im Main-Thread unter 3.                                  */
   ...
   Disable_UTM_UPIC (...);
   ...
   ExitThread(0);                                                  7.
}
```

1. Prozess und Main-Thread werden gestartet.
2. Starten zweier weiterer Threads über den entsprechenden Systemaufruf. Der Systemaufruf ist abhängig vom System und vom verwendeten Compiler.
   Jeder Thread wird mit der Funktion *UpicThread()* gestartet. In *UpicThread()* wird eine Conversation aufgebaut und bearbeitet (siehe 6.). *UpicThread* ist ein frei wählbarer Name.
3. Jeder Thread muss explizit einen *Enable_UTM_UPIC*- und einen *Disable_UTM_UPIC*-Aufruf ausführen. An dieser Stelle meldet sich der Main-Thread bei UPIC an. Nach dem *Enable_UTM_UPIC*-Aufruf können dann die CPI-C-Aufrufe für den Aufbau einer Conversation im Main-Thread und zum Bearbeiten dieser Conversation abgesetzt werden. Es können mehrere Conversations nacheinander im Main-Thread bearbeitet werden. Nach Beendigung der Conversation im Main-Thread muss sich dieser mit *Disable_UTM_UPIC* abmelden.
4. Der Main-Thread wartet, bis sich die beiden von ihm gestarteten Threads beenden.
5. Ende des Prozesses und des Main-Threads.
6. *UpicThread()* ist die Funktion, die aufgerufen wird, wenn ein neuer Thread gestartet wird. In ihr meldet sich der jeweilige Thread mit *Enable_UTM_UPIC* bei UPIC an und bearbeitet „seine Conversation“ (mit *Initialize_Conversation*, *Set_...*, *Send_Data*, *Receive* ...). Auch hier können mehrere Conversations nacheinander bearbeitet werden. Nach Beenden der letzten Conversation meldet sich der Thread mit *Disable_UTM_UPIC* ab.

   UpicThread() muss so programmiert werden, dass sich die nebeneinander laufenden Threads nicht gegenseitig beeinträchtigen. Der Code muss also so gestaltet werden, dass er gleichzeitig von mehreren Threads ausgeführt werden kann, d.h. die verwendeten Funktionen dürfen sich nicht gegenseitig den Kontext zerstören.
7. Ende des Threads.

Mit openUTM-Client wird die Source für ein Beispielprogramm zu „Multiple Conversations” ausgeliefert (siehe Abschnitt „Programmbeispiele für Windows-Systeme“ auf Seite 345).

4.8 DEFAULT-Server und DEFAULT-Name eines Client

In der Praxis ist es häufig so, dass ein Client hauptsächlich mit einem bestimmten UTM-Server kommuniziert. Um die Konfigurierung von UPIC-Clients und die Programmierung von CPI-C-Client-Programmen für diesen Fall zu vereinfachen, können Sie in der `upicfile` einen DEFAULT-Server für Ihre Client-Anwendung definieren (siehe Seite 304). Um mit dem DEFAULT-Server verbunden zu werden, kann das Client-Programm beim Initialisieren der Conversation mit *Initialize_Conversation* auf die Angabe eines Symbolic Destination Namens verzichten. Es übergibt einen leeren Namen an UPIC und wird dann automatisch mit dem DEFAULT-Server verbunden.

Sie können darüber hinaus einen Service am DEFAULT-Server als DEFAULT-Service definieren. Dazu geben Sie im Eintrag des DEFAULT-Server in der `upicfile` den Transaktionscode dieses Services an. Gibt das CPI-C-Programm dann beim Initialisieren einer Conversation zum DEFAULT-Server keinen Transaktionscode an (es ruft *Set_TP_Name* nicht auf), wird die Conversation automatisch zu dem DEFAULT-Service aufgebaut. Soll ein anderer Service am DEFAULT-Server gestartet werden, dann muss das Client-Programm mit *Set_TP_Name* den Transaktionscode dieses Service an UPIC übergeben (z.B. beim Vorgangs-Wiederanlauf muss *TP_name*=KDCDISP gewählt werden).

Ebenso können Sie in der `upicfile` einen DEFAULT-Namen für die lokale CPI-C-Client-Anwendung definieren. Gibt das Client-Programm beim Anmelden der Anwendung bei UPIC (mit *Enable_UTM_UPIC*) einen leeren lokalen Anwendungsnamen an, dann wird der Client mit dem DEFAULT-Namen bei UPIC angemeldet und UPIC verwendet die dem DEFAULT-Namen zugeordneten Adressinformationen zum Aufbau der Conversation.

Bei der Verwendung eines DEFAULT-Namens für die CPI-C-Anwendung kann es vorkommen, dass sich mehrere Programmläufe eines UPIC-Client zur gleichen Zeit mit demselben Namen bei einer UTM-Anwendung anmelden wollen. Das ist dann der Fall, wenn das Client-Programm mehrfach parallel gestartet wird oder ein Programm parallel mehrere Conversations zu einer UTM-Anwendung aufbauen will (Multiple Conversations). Damit diese Anmeldungen von der Server-Anwendung akzeptiert werden können, müssen die im folgenden Abschnitt beschriebenen Voraussetzungen erfüllt sein.

4.8.1 Mehrfachanmeldungen bei derselben UTM-Anwendung mit demselben Namen

Eine Client-Anwendung kann sich zu einem Zeitpunkt mehrfach mit demselben Namen an eine UTM-Anwendung anschließen.

Damit sich ein Client mehrfach mit demselben Namen anschließen kann, muss in der UTM-Server-Anwendung für den Rechner, an dem der Client abläuft, ein LTERM-Pool generiert sein, der die Mehrfachanmeldung unter demselben Namen unterstützt. Ein solcher LTERM-Pool wird bei openUTM wie folgt generiert:

TPOOL ...,CONNECT-MODE=MULTI

Für den Namen des Client, mit dem sich dieser bei der UTM-Anwendung anmeldet (PTERM-Name), darf in der UTM-Anwendung keine PTERM-Anweisung generiert sein (siehe openUTM-Handbuch „Anwendungen generieren“), sonst ist die Mehrfachanmeldung über den LTERM-Pool nicht möglich.

Das CPI-C-Programm kann sich über den LTERM-Pool maximal so oft an die UTM-Anwendung anschließen, wie LTERM-Partner im LTERM-Pool zur Verfügung stehen (die Anzahl wird durch die UTM-Administration eingestellt). Dabei kann es sich mit demselben, aber auch mit verschiedenen Namen anmelden.

4.9 CPI-C-Aufrufe bei UPIC

Im folgenden werden für jede Funktion die Ein- und Ausgabeparameter sowie die möglichen Returncodes beschrieben.

Allgemein gilt, dass sämtliche Parameter an der Schnittstelle per Adresse übergeben werden. Das Symbol → bzw. ← bedeutet, dass ein Parameter entweder ein Eingabe- oder ein Ausgabe-Parameter ist.

Die Länge für den *symbolic destination name* und die *Conversation_ID* ist immer genau acht Zeichen.

Die Returncodes, die an der Schnittstelle geliefert werden, sind unabhängig vom verwendeten Transportsystem. Eine Unterscheidung zwischen lokaler und remote Anbindung wird nur bei der Erklärung einiger Returncodes und bei den Hinweisen zu Fehlern vorgenommen.

Übersicht

Die Funktionen der Schnittstelle sind auf allen Plattformen in den Programmiersprachen C, C++ und COBOL nutzbar und stehen in Bibliotheken zur Verfügung.

Die folgende Beschreibung der CPI-C-Aufrufe ist aus diesem Grund so sprachunabhängig wie möglich gehalten. Sie benutzt jedoch die Notation der C-Schnittstelle. Im Abschnitt „COBOL-Schnittstelle“ auf Seite 246 sind Besonderheiten der COBOL-Schnittstelle beschrieben, die Sie beim Erstellen von CPI-C-Programmen in COBOL beachten müssen.

Die genaue Funktionsdeklaration wird für jeden Aufruf separat beschrieben.

Programmaufrufe

Ein Client kommuniziert mit einer UTM-Server-Anwendung, indem er Funktionen aufruft. Diese Aufrufe dienen dazu, die Characteristics für die Conversation festzulegen und Daten und Kontrollinformationen auszutauschen. Die von UPIC unterstützten CPI-C-Aufrufe können in zwei Gruppen eingeteilt werden:

- Starter-Set-Aufrufe
  Die Starter-Set-Aufrufe ermöglichen eine einfache Kommunikation mit einem UTM-Server. Sie dienen dem einfachen Austausch von Daten, z.B. übernehmen der initialisierten Werte für die Characteristic einer Conversation.

- Advanced Functions-Aufrufe
  Die Advanced Functions-Aufrufe ermöglichen zusätzliche Funktionen. Zum Beispiel können mit Set-Aufrufen die Conversation Characteristics modifiziert werden.

Funktionen aus dem Starter-Set

Funktion	Beschreibung
Initialize_Conversation	Conversation etablieren
Allocate	Conversation starten
Deallocate	Conversation abnormal beenden
Send_Data	Daten senden
Receive	Daten empfangen

Tabelle 8: Funktionen aus dem Starter-Set

Es wird davon ausgegangen, dass das CPI-C-Programm (Client) in jedem Fall der aktive Teil ist. Deshalb wird die CPI-C-Funktion *Accept_Conversation* nicht unterstützt.

Auf Systemen, die das Multi-Threading unterstützen, können in einem CPI-C-Programm zu einem Zeitpunkt mehrere Conversations zu verschiedenen UTM-Servern aktiv sein. Jede Conversation einschließlich zugehörigem *Enable_UTM_UPIC*- und *Disable_UTM_UPIC*-Aufruf muss in einem eigenen Thread ablaufen.

Auf allen anderen Systemen kann in einem CPI-C-Programmlauf zu einem Zeitpunkt nur **eine** Conversation aktiv sein.

Funktionen aus den Advanced Functions

Funktion	Beschreibung
Convert_Incoming	Empfangene Daten in lokalen Code konvertieren
Convert_Outgoing	Zu sendende Daten vom lokalem Code in den Code des Kommunikationspartners konvertieren
Deferred_Deallocate	Conversation beenden, sobald die laufende Transaktion erfolgreich beendet wurde
Extract_Conversation_State	Zustand der Conversation abfragen
Extract_Secondary_Information	Erweiterte Informationen abfragen
Extract_Partner_LU_Name	Wert der Conversation Characteristic *partner_LU_name* abfragen
Prepare_To_Receive	Die im Sendepuffer zwischengespeicherten Daten sofort an den Kommunikationspartner senden und in den Status "Receive" wechseln
Receive_Mapped_Data *	Daten zusammen mit Strukturierungsmerkmalen (Formatkennzeichen) empfangen
Send_Mapped_Data *	Daten zusammen mit Strukturierungsmerkmalen (Formatkennzeichen) senden
Set_Conversation_Security_Password	Passwort für eine UTM-Benutzerkennung setzen
Set_Conversation_Security_Type	Security-Funktionen aktivieren oder deaktivieren
Set_Conversation_Security_User_ID	UTM-Benutzerkennung setzen
Set_Partner_LU_name	Wert für die Conversation Characteristics *partner_LU_name* setzen
Set_Deallocate_Type	Werte für die Conversation Characteristic *deallocate_type* setzen
Set_Receive_Type	Werte für die Conversation Characteristic *receive_type* setzen
Set_Sync_Level	Werte für die Conversation Characteristic *sync_level* setzen
Set_TP_Name	Namen für ein Partnerprogramm setzen (Transaktionscode)

Tabelle 9: Advanced Functions

* Nicht Bestandteil von X/Open CPI-C Version 2

Zusätzliche Funktionen von UPIC

Funktion	**Beschreibung**
Enable_UTM_UPIC	Beim UPIC-Trägersystem anmelden
Extract_Client_Context	Client-Kontext ausgeben
Extract_Communication_Protocol	UPIC-Protokollversion abfragen
Extract_Conversation_Encryption_Level	Verschlüsselungsebene abfragen
Extract_Convertion	ASCII-EBCDIC-Konvertierung abfragen
Extract_Cursor_Offset	Offset der Cursor-Position abfragen
Extract_Secondary_Return_Code	Erweiterte Returncodes abfragen
Extract_Shutdown_State	Shutdown-Status des Servers abfragen
Extract_Shutdown_Time	Shutdown-Time des Servers abfragen
Extract_Transaction_State	Vorgangs- und Transaktionsstatus des Servers abfragen
Disable_UTM_UPIC	Beim UPIC-Trägersystem abmelden
Set_Allocate_Timer	Timer für Allocate setzen
Set_Client_Context	Client-Kontext setzen
Set_Communication_Protocol	UPIC-Protokollversion setzen
Set_Convertion	ASCII-EBCDIC-Konvertierung setzen
Set_Conversation_Encryption_Level	Verschlüsselungsebene setzen
Set_Conversation_Security_New_ Password	Neues Passwort für eine UTM-Benutzerkennung setzen
Set_Function_Key	Wert der zu übertragenden Funktionstaste setzen
Set_Receive_Timer	Timeout Timer für den blockierenden Empfang von Daten setzen
Set_Partner_Host_Name	Hostname der Partner-Anwendung setzen
Set_Partner_IP_Address	IP-Adresse der Partner-Anwendung setzen
Set_Partner_Port	TCP/IP-Port der Partner-Anwendung setzen
Set_Partner_Tsel	TSEL der Partner-Anwendung setzen
Set_Partner_Tsel_Format	TSEL-Format der Partner-Anwendung setzen
Specify_Local_Tsel	TSEL der lokalen Anwendung setzen
Specify_Local_Tsel_Format	TSEL-Format der lokalen Anwendung setzen
Specify_Local_Port	TCP/IP-Port der lokalen Anwendung setzen
Specify_Secondary_Return_Code	Eigenschaften des erweiterten Returncodes setzen

Tabelle 10: Zusätzliche Funktionen von UPIC

Allocate - Conversation einrichten

Der Aufruf *Allocate* (CMALLC) richtet für ein Programm eine Conversation zu einem UTM-Vorgang ein. Der Name des CPI-C-Programms wurde beim vorhergehenden *Enable_UTM_UPIC*-Aufruf angegeben.

Syntax

CMALLC (conversation_ID, return_code)

Parameter

→ conversation_ID	Identifikation der bereits initialisierten Conversation (wird vom Initialize-Aufruf geliefert)
← return_code	Ergebnis des Funktionsaufrufs

Ergebnis (*return_code*)

CM_OK

Aufruf ok

CM_ALLOCATE_FAILURE_RETRY

UPIC-L Die Conversation kann aufgrund eines vorübergehenden Betriebsmittelengpasses nicht eingerichtet werden. Überprüfen Sie auch die Fehlermeldung der lokalen UTM-Anwendung.

CM_ALLOCATE_FAILURE_NO_RETRY

Mögliche Ursachen:

- Die Conversation kann aufgrund eines Fehlers nicht eingerichtet werden, z.B. die Transportverbindung zur UTM-Anwendung konnte nicht aufgebaut werden.
- Die Transportverbindung wurde von UTM-Seite zurückgewiesen, weil in der UTM-Anwendung ein TPOOL oder PTERM-Anschlusspunkt mit ENCRYPTION_LEVEL=1 (oder 2, 3, 4) definiert wurde, aber das Zusatzprodukt openUTM-CRYPT ist nicht installiert.
- Die Transportverbindung wurde von UTM-Seite zurückgewiesen, weil in der UTM-Anwendung ein TPOOL oder PTERM-Anschlusspunkt mit ENCRYPTION_LEVEL=NONE definiert wurde, der aufgerufene TAC wurde mit ENCRYPTION_LEVEL=1 (oder 2) definiert, aber das Zusatzprodukt openUTM-CRYPT ist nicht installiert.

CM_OPERATION_INCOMPLETE

Der Aufruf wurde durch den Ablauf des Timers, der mit Set_Allocate_Timer gesetzt wurde, unterbrochen.

CM_PARAMETER_ERROR
Weder in der `upicfile` noch mit einem *Set_TP_Name*-Aufruf wurde ein TAC angegeben oder *conversation_security_type* ist CM_SECURITY_PROGRAM und die Characteristic *security_user_ID* ist nicht gesetzt.

CM_PROGRAM_STATE_CHECK
Der Aufruf ist im aktuellen Zustand nicht erlaubt.

CM_PROGRAM_PARAMETER_CHECK
Der Wert für *conversation_ID* ist ungültig.

CM_PRODUCT_SPECIFIC_ERROR
- Es handelt sich um einen Protokollfehler.
- Bei TNS-losem Betrieb ist der Hostname (der über *Set_Partner_Host_Name* oder in der `upicfile` angegeben wurde) nicht in der Datei `hosts` definiert.
- In der `upicfile` ist für diese Conversation ein RSA-Schlüssel hinterlegt, der sich vom empfangenen RSA-Schlüssel in Inhalt oder Länge unterscheidet.

CM_SECURITY_NOT_SUPPORTED
- Die Partner-Anwendung kann den gewünschten *security_type* nicht unterstützen.
- Es wurde ein neues Passwort gesetzt, aber die Partner-Anwendung, zu der eine Conversation aufgebaut wurde, unterstützt keine Passwortänderungen vom UPIC-Client aus.

Zustandsänderung

- Falls das Ergebnis CM_OK ist, wird die Conversation etabliert und das Programm geht in den Zustand "Send" über.
- Falls das Ergebnis CM_ALLOCATE_FAILURE_RETRY/NO_RETRY oder CM_SECURITY_NOT_SUPPORTED ist, geht das Programm in den Zustand "Reset" über.
- In allen anderen Fehlersituationen ändert das Programm seinen Zustand nicht.

Hinweis

- Lehnt die UTM-Anwendung den Vorgangsstart z.B. wegen ungültigem Transaktionscode ab, wird dies erst beim nächsten *Receive*-Aufruf zurückgemeldet.
- Falls die angegebene Benutzerkennung bei der UTM-Anwendung nicht generiert wurde oder falls für eine generierte Benutzerkennung ein falsches oder gar kein Passwort geschickt wurde, so wird dies erst beim nächsten *Receive*-Aufruf zurückgemeldet.

Verhalten im Fehlerfall

CM_ALLOCATE_FAILURE_RETRY
: Vorübergehender Betriebsmittelengpass bei der Kommunikation. Erst *Initialize_Conversation*, dann den *Allocate*-Aufruf wiederholen.

CM_ALLOCATE_FAILURE_NO_RETRY
: Eventuell UTM-Anwendung hochfahren oder das beim *Enable_UTM_UPIC* angegebene PTERM bei openUTM generieren. Eventuell müssen Sie auch das Verschlüsselungsmodul installieren oder die Verschlüsselungsebene ändern.

CM_PARAMETER_ERROR
: Eintrag für den aktuellen *sym_dest_name* um einen TAC erweitern oder TAC mit einem *Set_TP_Name*-Aufruf angeben.

CM_PROGRAM_STATE_CHECK
: Programm ändern.

CM_PROGRAM_PARAMETER_CHECK
: Programm ändern.

CM_PRODUCT_SPECIFIC_ERROR
- Keinen oder den gültigen RSA-Schlüssel hinterlegen.
- Systemdienst informieren und Diagnoseunterlagen erstellen

Funktionsdeklaration: Allocate

```
CM_ENTRY Allocate ( unsigned char CM_PTR  conversation_ID,
                    CM_RETURN_CODE CM_PTR  return_code)
```

Convert_Incoming - Konvertieren vom Code des Senders in lokalen Code

X/W X/W X/W Beim Trägersystem UPIC auf Unix-Systemen und Windows-Systemen konvertiert der Aufruf *Convert_Incoming* (CMCNVI) die Daten von EBCDIC in den lokal auf der Maschine verwendeten Code.

B B Beim Trägersystem UPIC im BS2000/OSD konvertiert *Convert_Incoming* die Daten von ASCII in den lokal auf dem BS2000-Rechner verwendeten Code.

Syntax

CMCNVI (data, length, return_code)

Parameter

↔ data	Adresse der Daten, die konvertiert werden sollen. Der Dateninhalt wird durch die konvertierten Daten überschrieben.
→ length	Länge der Daten, die konvertiert werden
← return_code	Ergebnis des Funktionsaufrufs

Ergebnis (*return_code*)

CM_OK
    Aufruf ok

Zustandsänderung

Dieser Aufruf ändert den Zustand des Programms nicht.

Hinweis

- Die Daten müssen in abdruckbarer Form vorliegen.
- Die verwendete Konvertierungstabelle finden Sie:
  - X/W – auf Unix- und Windows-Systemen in der Datei *upic-dir*\kcsaeea.c
  - B – auf BS2000-Systemen in der Bibliothek *$userid*.SYSLIB.UTM-CLIENT.061

Funktionsdeklaration: Convert_Incoming

```
CM_ENTRY Convert_Incoming ( unsigned char CM_PTR  string,
                                   CM_INT32 CM_PTR  string_length,
                               CM_RETURN_CODE CM_PTR  return_code)
```

Convert_Outgoing - Konvertieren von lokalem Code in den Code des Empfängers

X/W Beim Trägersystem UPIC auf Unix-Systemen und Windows-Systemen konvertiert der Aufruf *Convert_Outgoing* (CMCNVO) die Daten vom lokal auf der Maschine verwendeten Code nach EBCDIC.



Beim Trägersystem UPIC im BS2000 konvertiert *Convert_Outgoing* die Daten vom lokal verwendeten Code immer in ASCII.

Syntax

CMCNVO (data, length, return_code)

Parameter

↔ data	Adresse der Daten, die konvertiert werden sollen. Der Dateninhalt wird durch die konvertierten Daten überschrieben.
→ length	Länge der Daten, die konvertiert werden
← return_code	Ergebnis des Funktionsaufrufs

Ergebnis (*return_code*)

CM_OK
    Aufruf ok

Zustandsänderung

Dieser Aufruf ändert den Zustand des Programms nicht.

Hinweis

- Die Daten müssen in abdruckbarer Form vorliegen.
- Die verwendete Konvertierungstabelle finden Sie:
  - X/W auf Unix- und Windows-Systemen in der Datei *upic-dir*\kcsaeea.c
  - B auf BS2000-Systemen in der Bibliothek *$userid*.SYSLIB.UTM-CLIENT.061

Funktionsdeklaration: Convert_Outgoing

```
CM_ENTRY Convert_Outgoing ( unsigned char CM_PTR  string,
                                   CM_INT32 CM_PTR  string_length,
                             CM_RETURN_CODE CM_PTR  return_code)
```

Deallocate - Conversation beenden

Mit dem Aufruf *Deallocate* (CMDEAL) wird die Conversation vom CPI-C-Programm abnormal beendet. Nach Ausführung des Aufrufs ist die *conversation_ID* keiner Conversation mehr zugeordnet. Im Normalfall wird eine Conversation immer mit dem UTM-Vorgang beendet. Eine Beendigung der Conversation durch das CPI-C-Programm gilt immer als abnormale Beendigung. Deshalb muss, bevor ein *Deallocate*-Aufruf gemacht wird, mit der Funktion *Set_Deallocate_Type* (CMSDT) der Wert für *deallocate_type* auf CM_DEALLOCATE_ABEND gesetzt werden.

Syntax

CMDEAL (conversation_ID, return_code)

Parameter

→ conversation_ID	Identifikation der Conversation, die beendet werden soll.
← return_code	Ergebnis des Funktionsaufrufs

Ergebnis (*return_code*)

CM_OK
: Aufruf ok

CM_PROGRAM_STATE_CHECK
: Der Aufruf ist im aktuellen Zustand nicht erlaubt.

CM_PROGRAM_PARAMETER_CHECK
: Der Wert für *conversation_ID* ist ungültig.

CM_PRODUCT_SPECIFIC_ERROR
: Der Wert für *deallocate_type* ist nicht durch einen vorangegangenen *Set_Deallocate_Type* Aufruf auf CM_DEALLOCATE_ABEND gesetzt.

Zustandsänderung

Falls das Ergebnis CM_OK ist, geht das Programm in den Zustand "Reset" über. Bei allen anderen Fehlersituationen ändert das Programm seinen Zustand nicht.

Verhalten im Fehlerfall

CM_PROGRAM_STATE_CHECK
: Programm ändern.

CM_PROGRAM_PARAMETER_CHECK
: Programm ändern.

CM_PRODUCT_SPECIFIC_ERROR
: Programm ändern und *Set_Deallocate_Type*-Aufruf einbauen.

Funktionsdeklaration: Deallocate

```
CM_ENTRY Deallocate ( unsigned char CM_PTR  conversation_ID,
                      CM_RETURN_CODE CM_PTR  return_code)
```

Deferred_Deallocate - Conversation nach Transaktionsende beenden

Mit dem Aufruf *Deferred_Deallocate* (CMDFDE) wird die Conversation vom CPI-C-Programm beendet, sobald die laufende Transaktion erfolgreich beendet ist. Der Aufruf darf innerhalb einer Transaktion zu jedem Zeitpunkt aufgerufen werden. *Deferred_Deallocate* dient nur der besseren Portierbarkeit von CPI-C-Programmen. Er ändert den Zustand des Programms nicht.

Syntax

CMDFDE (conversation_ID, return_code)

Parameter

→ conversation_ID	Identifikation der Conversation, die beendet werden soll.
← return_code	Ergebnis des Funktionsaufrufs

Ergebnis (*return_code*)

CM_OK
: Aufruf ok

CM_PROGRAM_PARAMETER_CHECK
: Der Wert für *conversation_ID* ist ungültig.

CM_PROGRAM_STATE_CHECK
: Das Programm ist im Zustand „Start".

CM_PRODUCT_SPECIFIC_ERROR
: Die UPIC-Instanz konnte nicht gefunden werden.

Zustandsänderung

Das Programm ändert seinen Zustand nicht.

Verhalten im Fehlerfall

CM_PROGRAM_PARAMETER_CHECK
: Programm ändern.

CM_PRODUCT_SPECIFIC_ERROR
: Das Betriebssystem kann nicht genügend Speicherplatz für interne Puffer bereitstellen. Überprüfen Sie Ihr Programm auf zu hohe Speicherplatzanforderung und starten Sie ggf. Ihr System neu.

CM_PROGRAM_STATE_CHECK
: Programm ändern.

Funktionsdeklaration: Deferred_Deallocate

```
CM_ENTRY Deferred_Deallocate ( unsigned char CM_PTR  conversation_ID,
                               CM_RETURN_CODE CM_PTR  return_code)
```

Disable_UTM_UPIC - Vom Trägersystem UPIC abmelden

Mit dem Aufruf *Disable_UTM_UPIC* (CMDISA) meldet sich ein Programm vom UPIC-Trägersystem ab. Nach erfolgreicher Ausführung des Aufrufs sind keine weiteren CPI-C-Aufrufe erlaubt. Falls es für das Programm noch eine Verbindung gibt, wird diese abgebaut. Außerdem wird die Abmeldung vom Transportsystem durchgeführt.

Dieser Aufruf muss der letzte Aufruf eines CPI-C-Programmes sein. Er ist nicht nötig, wenn Sie nach dem Beenden einer Conversation mit einem weiteren *Initialize*-Aufruf fortfahren.

Diese Funktion gehört zu den zusätzlichen Funktionen von UPIC, sie ist nicht Bestandteil der CPI-C-Schnittstelle.

Syntax

CMDISA (local_name, local_name_length, return_code)

Parameter

→ local_name — Name des Programms, d.h. der Name, der bei dem vorangegangenen *Enable_UTM_UPIC*-Aufruf angegeben wurde.

→ local_name_length — Länge von *local_name*

Minimum: 0, Maximum: 8

local_name_length=0 bedeutet, dass ein „leerer lokaler Anwendungsname“ übergeben wird (siehe Abschnitt „Enable_UTM_UPIC - Beim Trägersystem UPIC anmelden“ auf Seite 111)

← return_code — Ergebnis des Funktionsaufrufs

Ergebnis (*return_code*)

CM_OK
: Aufruf ok

CM_PROGRAM_STATE_CHECK
: Der Aufruf ist im aktuellen Zustand nicht erlaubt.

CM_PROGRAM_PARAMETER_CHECK
: Das Programm ist nicht mit *local_name* an UPIC angemeldet, oder der Wert für *local_name_length* ist < 1 oder > 8.

CM_PRODUCT_SPECIFIC_ERROR
: Beim Abmelden von UPIC oder beim Abbau der Verbindung ist ein Fehler aufgetreten.

Zustandsänderung

Falls das Ergebnis CM_OK ist, wurde das Programm abgemeldet und geht in den Zustand "Start" über. In allen anderen Fällen ändert das Programm seinen Zustand nicht.

Hinweis

Den Aufruf müssen Sie auch dann verwenden, wenn Sie bei einer Fehlersituation im Anwendungsprogramm den Prozess mit *exit()* beenden wollen.
Aus Performancegründen sollte diese Funktion, falls kein Fehler auftritt, nur unmittelbar vor der Prozessbeendigung aufgerufen werden!

Verhalten im Fehlerfall

CM_PRODUCT_SPECIFIC_ERROR
: Systemdienst informieren und Diagnoseunterlagen erstellen.

CM_PROGRAM_STATE_CHECK
: Programm ändern.

CM_PROGRAM_PARAMETER_CHECK
: Programm ändern.

Funktionsdeklaration: Disable_UTM_UPIC

```
CM_ENTRY Disable_UTM_UPIC ( unsigned char CM_PTR  local_name,
                                  CM_INT32 CM_PTR  local_name_length,
                            CM_RETURN_CODE CM_PTR  return_code)
```

Enable_UTM_UPIC - Beim Trägersystem UPIC anmelden

Dieser Aufruf muss gemacht werden, bevor andere CPI-C-Aufrufe verwendet werden. Mit dem Aufruf *Enable_UTM_UPIC* (CMENAB) meldet sich ein Programm mit seinem Namen beim UPIC-Trägersystem an. Der Name dient dazu, die Verbindung zwischen UTM-Service und CPI-C-Programm aufzubauen (siehe auch Abschnitt „Initialize_Conversation - Conversation Characteristics initialisieren“ auf Seite 147).

In der `upicfile` können Sie einen DEFAULT-Namen für die CPI-C-Anwendung definieren (LN.DEFAULT-Eintrag; siehe Abschnitt „Side Information für die lokale Anwendung“ auf Seite 312). Wenn sich das CPI-C-Programm mit diesem DEFAULT-Namen beim Trägersystem UPIC anmelden soll, dann kann es im Feld *local_name* einen „leeren lokalen Anwendungsnamen“ übergeben. UPIC sucht dann in der `upicfile` nach dem LN.DEFAULT-Eintrag und verwendet den zugehörigen Anwendungsnamen zum Verbindungsaufbau mit dem UTM-Service. Es können sich gleichzeitig mehrere CPI-C-Programmläufe mit dem DEFAULT-Namen anmelden und auch Conversations zu demselben UTM-Service aufbauen.

Nach erfolgreicher Ausführung des *Enable_UTM_UPIC*-Aufrufs ist eine intakte Ablaufumgebung für das Programm bereitgestellt. Nach diesem Aufruf bleiben Änderungen in der `upicfile` bis zum nächsten *Enable_UTM_UPIC*-Aufruf für das Programm unwirksam.

Diese Funktion gehört zu den zusätzlichen Funktionen des Trägersystems UPIC; sie ist nicht Bestandteil der CPI-C-Schnittstelle.

Syntax

CMENAB (local_name, local_name_length, return_code)

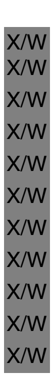
Parameter

→ local_name

Name des Programms.
Folgende Angaben sind möglich (siehe auch Abschnitt „Side Information für die lokale Anwendung“ auf Seite 312):

bei UPIC-Remote:

- lokaler Anwendungsname, der in der `upicfile` definiert ist.
- Name, mit dem das Programm im TNS-Directory eingetragen bzw. bei CMX bekannt ist.
- beliebiger Name, dessen TNS-Eigenschaften mit nachfolgenden *Specify*-Aufrufen noch verändert werden können.
- leerer lokaler Anwendungsname.
  Das Programm meldet sich dann mit dem DEFAULT-Namen der CPI-C-Anwendung bei UPIC an. Voraussetzung ist, dass zum Zeitpunkt des Aufrufs in der `upicfile` ein LN.DEFAULT-Eintrag existiert.



bei UPIC-Local:

- PTERM-Name, unter dem der Client in der Konfiguration der UTM-Anwendung bekannt ist.
- lokaler Anwendungsname, der in der `upicfile` definiert ist.
- Existiert in der UTM-Partner-Anwendung ein LTERM-Pool für den Partnertyp UPIC-L (TPOOL mit PTYPE=UPIC-L), dann können Sie für *local_name* einen beliebigen, bis zu 8 Zeichen langen Namen angeben.
- leerer lokaler Anwendungsname
  Voraussetzung ist, dass zum Zeitpunkt des Aufrufs in der `upicfile` ein LN.DEFAULT-Eintrag existiert.

Einen leeren lokalen Anwendungsnamen können Sie übergeben, indem Sie:

- in *local_name* 8 Blanks übergeben und *local_name_length*=8 setzen
- *local_name_length*=0 setzen.

Übergeben Sie einen leeren lokalen Anwendungsnamen, dann übernimmt UPIC den Anwendungsnamen des LN.DEFAULT-Eintrags, um die Verbindung zur UTM-Partner-Anwendung aufzubauen.

→ local_name_length
Länge von *local_name*
Minimum: 0, Maximum: 8

Wird in *local_name* ein lokaler Anwendungsname aus der `upicfile` eingetragen, dann muss *local_name_length*=8 angegeben werden.

Geben Sie *local_name_length*=0 an, dann wird der Inhalt des Feldes *local_name* ignoriert, d.h. *local_name* wird als „leerer lokaler Anwendungsname“ behandelt. In der `upicfile` muss ein LN.DEFAULT-Eintrag existieren.

← return_code
Ergebnis des Funktionsaufrufs

Ergebnis (*return_code*)

CM_OK
Aufruf ok

CM_PROGRAM_STATE_CHECK
Das Programm ist bereits an UPIC angemeldet.

CM_PROGRAM_PARAMETER_CHECK
mögliche Ursachen:
- Der Wert für *local_name_length* ist kleiner als 0 oder größer als 8.
- Es ist nicht genügend interner Speicher vorhanden oder
- ein Zugriff auf die `upicfile` ist fehlgeschlagen.

CM_PRODUCT_SPECIFIC_ERROR
mögliche Ursachen:
- Die UPIC-Instanz konnte nicht gefunden werden oder



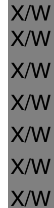
- nur bei UPIC-Local auf Unix- und Windows-Systemen: die Umgebungsvariable UTMPATH ist nicht gesetzt.

Zustandsänderung

Falls das Ergebnis CM_OK ist, geht das Programm in den Zustand "Reset" über. In allen anderen Fällen ändert das Programm seinen Zustand nicht.

Hinweis

- Es können sich gleichzeitig mehrere CPI-C-Programmläufe mit demselben Namen beim Trägersystem UPIC anmelden.
- Ein mehrfach gestartetes CPI-C-Programm kann sich auch mehrfach mit demselben Namen bei derselben UTM-Anwendung anschließen (z.B. der Anwendungsname, der dem DEFAULT-Namen zugeordnet ist). Dazu muss die UTM-Anwendung folgendermaßen konfiguriert sein:
  - Es darf kein LTERM-Partner explizit für diesen openUTM-Client generiert sein, d.h. es darf kein PTERM mit seinem Namen und PTYPE=UPIC-R für diesen Rechner in der Konfiguration der UTM-Anwendung existieren.
  - Für den Rechner, an dem der Client abläuft, ist ein LTERM-Pool (TPOOL) mit CONNECT-MODE=MULTI generiert. Das CPI-C-Programm kann sich unter demselben Namen dann maximal so oft an die UTM-Anwendung anschließen, wie LTERM-Partner im LTERM-Pool zur Verfügung stehen (die Anzahl wird durch die UTM-Administration eingestellt).



- bei *UPIC-Local*:
  Damit sich das CPI-C-Programm bei der lokalen UTM-Anwendung anmelden kann, muss die Umgebungsvariable UTMPATH gesetzt sein.
  In seltenen Fällen kann es bei lokaler Kommunikation geschehen, dass sich die Funktion mit CM_PROGRAM_STATE_CHECK beendet, obwohl kurz zuvor *Disable_UTM_UPIC* aufgerufen wurde und CM_OK zurücklieferte. Die Ursache ist ein unvollständiger openUTM-interner Verbindungsabbau.

Verhalten im Fehlerfall

CM_PRODUCT_SPECIFIC_ERROR

- Das Betriebssystem kann nicht genügend Speicherplatz für interne Puffer bereitstellen; prüfen Sie Ihr Programm auf zu hohe Speicherplatzanforderung und starten Sie ggf. Ihr System neu.
- *Bei UPIC-Local*:
  Die Umgebungsvariable UTMPATH setzen und das Programm neu starten.



CM_PROGRAM_STATE_CHECK

Programm ändern.

CM_PROGRAM_PARAMETER_CHECK

- Programm ändern.
- u.U. virtuellen Speicher vergrößern

Funktionsdeklaration: Enable_UTM_UPIC

```
CM_ENTRY Enable_UTM_UPIC ( unsigned char CM_PTR  local_name,
                                CM_INT32 CM_PTR  local_name_length,
                          CM_RETURN_CODE CM_PTR  return_code)
```

Extract_Client_Context - Client-Kontext abfragen

Mit dem Aufruf *Extract_Client_Context* erhält ein Programm den Client-spezifischen Kontext, den openUTM als letztes gesendet hat.

Der Kontext wird von openUTM bis zum Ende der Conversation gesichert, falls er nicht durch einen neuen Kontext überschrieben wird. Wird vom Client ein Wiederanlauf angefordert, so wird der zuletzt gesicherte Kontext zusammen mit der letzten Dialog-Nachricht an den Client zurück übertragen.

Der Client-Kontext wird von openUTM nur gesichert, wenn eine UTM-Benutzerkennung mit Restartfunktionalität angemeldet ist, da nur in diesem Fall ein Vorgangs-Wiederanlauf möglich ist.

Der Aufruf *Extract_Client_Context* ist im Zustand "Send" und "Receive" und im Zustand "Reset" unmittelbar nach einem *Receive*-/*Receive_Mapped_Data*-Aufruf erlaubt.

Extract_Client_Context ist nicht Bestandteil der CPI-C Spezifikation, sondern eine zusätzliche Funktion des UPIC-Trägersystems.

Syntax

CMECC (conversation_ID, buffer, requested_length, data_received, received_length, return_code)

Parameter

→ conversation_ID
: Identifikation der bereits initialisierten Conversation (wird vom *Initialize*-Aufruf geliefert).

← buffer
: Puffer, in dem die Daten empfangen werden.
Falls der Wert von *received_length* = 0 ist, ist der Inhalt von *buffer* undefiniert.

→ requested_length
: Maximale Länge der Daten, die empfangen werden können.

← data_received
: Gibt an, ob das Programm den Client-Kontext vollständig empfangen hat.

Falls das Ergebnis (*return_code*) nicht den Wert CM_OK hat, ist der Wert von *data_received* undefiniert.

data_received kann folgende Werte annehmen:

CM_COMPLETE_DATA_RECEIVED
Der Client-Kontext wurde vollständig empfangen.

CM_INCOMPLETE_DATA_RECEIVED
Der Client-Kontext ist nicht vollständig vom Programm empfangen worden.

← received_length Länge der empfangenen Daten. Ist der Wert von *received_length* = 0, so liegt kein Client-Kontext vor. Der Wert von *received_length* ist undefiniert, falls das Ergebnis (*return_code*) nicht den Wert CM_OK hat.

← return_code Ergebnis des Funktionsaufrufs.

Ergebnis (*return_code*)

CM_OK
Aufruf OK

CM_CALL_NOT_SUPPORTED
Die Funktion wird nicht unterstützt. Dieser Returncode tritt auf, wenn kein Client-Kontext verwendet werden kann, da die UTM-Partner-Anwendung mit Version < 5.0 dies nicht unterstützen kann.

CM_PROGRAM_PARAMETER_CHECK
Der Wert in *conversation_ID* ist ungültig oder der Wert für *requested_length* ist größer als 32767 oder kleiner 1.

Der Wert der *conversation_ID* ist ungültig, weil die Funktion nach Ende der Conversation mehr als einmal aufgerufen wurde oder weil noch keine Conversation existierte (nach dem *Enable_UTM_UPIC*-Aufruf ist noch kein *Initialize_Conversation*-Aufruf erfolgt).

CM_PRODUCT_SPECIFIC_ERROR
Die UPIC-Instanz konnte nicht gefunden werden.

CM_PROGRAM_STATE_CHECK
Die Conversation ist nicht im Zustand "Reset", "Send" oder "Receive".

Hinweis

- Falls eine Teilnachricht mit *Receive*-/*Receive_Mapped_Data*-Aufruf(en) empfangen wurde (*data_received* hat den Wert CM_COMPLETE_DATA_RECEIVED), so werden die Parameter *client_context* und *client_context_length* bei einem nachfolgenden *Receive*-/*Receive_Mapped_Data*-Aufruf zurückgesetzt.
- Der Wert der *conversation_ID* bleibt für diesen Funktionsaufruf nach dem Ende einer Conversation so lange gültig, bis ein *Initialize_Conversation*- oder ein *Extract_Client_Context*-Aufruf erfolgt ist.
- Der interne Puffer besitzt eine beschränkte Grösse von derzeit 8 Byte.

- openUTM sendet derzeit immer einen Client Context der Länge 8 Byte zurück. D.h., wenn von UPIC ein gültiger Client-Kontext empfangen worden ist, so hat *received_length* die Länge 8.
  Falls an openUTM ein Client-Kontext mit einer Länge < 8 Byte gesendet worden ist, dann wird der Client-Kontext von openUTM mit binär 0 auf die Länge 8 aufgefüllt.
- Ist der Wert für *requested_length* kleiner als die Länge des intern gespeicherten *client_context*, so wird der vom Anwendungsprogramm zur Verfügung gestellte Puffer vollständig gefüllt und *data_received* auf CM_INCOMPLETE_DATA_RECEIVED gesetzt. Folgt unmittelbar ein weiterer CMECC-Aufruf mit einem genügend großem Wert für *requested_length* (d.h. ≥ 8), so wird der Puffer mit einem solchen Aufruf komplett gelesen.

Verhalten im Fehlerfall

CM_CALL_NOT_SUPPORTED
: Ist nicht unbedingt ein Fehler des Programms. Falls eine UPIC-R Anwendung mit verschiedenen UTM-Partner-Anwendungen kommuniziert, bedeutet dieser Returncode lediglich, dass die Anwendung mit einer UTM-Anwendung kommuniziert, die keinen Client-Kontext senden kann. Das Programm kann sich diesen Returncode merken und auf weitere Aufrufe bzgl. Client-Kontext verzichten.

CM_PROGRAM_STATE_CHECK
: Programm ändern.

CM_PROGRAM_PARAMETER_CHECK
: Programm ändern.

CM_PRODUCT_SPECIFIC_ERROR
: Das Betriebssystem kann nicht genügend Speicherplatz für interne Puffer bereitstellen. Überprüfen Sie Ihr Programm auf zu hohe Speicherplatzanforderung und starten Sie ggf. Ihr System neu.

Funktionsdeklaration: Extract_Client_Context

```
CM_ENTRY Extract_Client_Context (
                        unsigned char         CM_PTR   conversation_ID,
                        unsigned char         CM_PTR   buffer,
                        CM_INT32              CM_PTR   requested_length,
                        CM_DATA_RECEIVED_TYPE CM_PTR   data_received,
                        CM_INT32              CM_PTR   received_length,
                        CM_RETURN_CODE        CM_PTR   return_code )
```

Extract_Communication_Protocol – Zustand der Conversation Characteristic PROTOCOL abfragen

Mit dem Aufruf *Extract_Communication_Protocol* (CMECP) erhält ein Programm den aktuellen Wert für die Characteristic *PROTOCOL* der Conversation.

Diese Funktion gehört zu den zusätzlichen Funktionen des Trägersystems UPIC; sie ist nicht Bestandteil der CPI-C-Schnittstelle.

Der Aufruf *Extract_Communication_Protocol* ist nur im Zustand "Init" erlaubt.

Syntax

CMECP (conversation_ID, protocol, return_code)

Parameter

→ conversation_ID Identifikation der Conversation

← protocol der Wert gibt an, welches Kommunikationsprotokoll zwischen UPIC-Client und UTM-Server benutzt wird.

Für *protocol* können folgende Werte zurückgegeben werden:

CM_COMMUNICATION_PROTOCOL_34
Die Kommunikation zwischen UPIC-Client und UTM-Server erfolgt auf Basis des Protokolls 34.

CM_COMMUNICATION_PROTOCOL_40
Die Kommunikation zwischen UPIC-Client und UTM-Server erfolgt auf Basis des Protokolls 40.

← return_code Ergebnis des Funktionsaufrufes

Ergebnis (*return_code*)

CM_OK
Aufruf OK

CM_PROGRAM_PARAMETER_CHECK
Der Wert in *conversation_ID* ist ungültig.

CM_PRODUCT_SPECIFIC_ERROR
Die UPIC-Instanz konnte nicht gefunden werden.

CM_PROGRAM_STATE_CHECK
Die Conversation ist nicht im Zustand “Initialize”.

Zustandsänderung

Der Aufruf ändert den Zustand der Conversation nicht.

Hinweis

Falls der Returncode von CM_OK verschieden ist, bleibt die Characteristic unverändert.

Verhalten im Fehlerfall

CM_PROGRAM_STATE_CHECK
: Programm ändern

CM_PROGRAM_PARAMETER_CHECK
: Programm ändern

CM_PRODUCT_SPECIFIC_ERROR
: Das Betriebssystem kann nicht genügend Speicherplatz für interne Puffer bereitstellen. Überprüfen Sie Ihr Programm auf zu hohe Speicherplatzanforderung und starten Sie ggf. Ihr System neu.

Funktionsdeklaration: Extract_Communication_Protocol

```
CM_ENTRY Extract_Communication_Protocol(
                unsigned char               CM_PTR  conversation_ID,
                CM_COMMUNICATION_PROTOCOL   CM_PTR  protocol,
                CM_RETURN_CODE              CM_PTR  return_code )
```

Extract_Conversation_Encryption_Level - Verschlüsselungsebene abfragen

Mit dem Aufruf *Extract_Conversation_Encryption_Level* (CMECEL) erhält ein Programm die eingestellte Verschlüsselungsebene der Conversation.

Der Aufruf *Extract_Conversation_Encryption_Level* ist im Zustand "Initialize", "Send" und "Receive" erlaubt.



UPIC-Local: Die Datenübertragung ist durch die Art der Übertragung selbst geschützt. Der Aufruf *Extract_Conversation_Encryption_Level* wird nicht unterstützt.

Diese Funktion gehört zu den zusätzlichen Funktionen des Trägersystems UPIC; sie ist nicht Bestandteil der CPI-C-Schnittstelle.

Syntax

CMECEL (conversation_ID, encryption_level, return_code)

Parameter

→ conversation_ID Identifikation der Conversation

← encryption_level Folgende Werte können Sie erhalten:

CM_ENC_LEVEL_NONE
Die Benutzerdaten der Conversation werden unverschlüsselt übertragen.

CM_ENC_LEVEL_1
Die Benutzerdaten werden verschlüsselt übertragen, zum Verschlüsseln wird der DES-Algorithmus benutzt. Für den Austausch des DES-Schlüssels wird ein RSA-Schlüssel mit einer Schlüssellänge von 200 bit verwendet.

CM_ENC_LEVEL_2
Die Benutzerdaten werden verschlüsselt übertragen, zum Verschlüsseln wird der AES-Algorithmus benutzt. Für den Austausch des AES-Schlüssels wird ein RSA-Schlüssel mit einer Schlüssellänge von 512 bit verwendet.

CM_ENC_LEVEL_3
Die Benutzerdaten werden verschlüsselt übertragen, zum Verschlüsseln wird der AES-Algorithmus benutzt. Für den Austausch des AES-Schlüssels wird ein RSA-Schlüssel mit einer Schlüssellänge von 1024 bit verwendet.

CM_ENC_LEVEL_4
Die Benutzerdaten werden verschlüsselt übertragen, zum Verschlüsseln wird der AES-Algorithmus benutzt. Für den Austausch des AES-Schlüssels wird ein RSA-Schlüssel mit einer Schlüssellänge von 2048 bit verwendet.

← return_code Ergebnis des Funktionsaufrufs

Ergebnis (*return_code*)

CM_OK
Aufruf ok



CM_CALL_NOT_SUPPORTED
Die Funktion wird nicht unterstützt. Dieser Returncode tritt nur bei UPIC-L auf. Er zeigt dem Programm an, dass keine Verschlüsselung notwendig ist.

CM_PROGRAM_STATE_CHECK
Die Conversation ist im Zustand "Start" oder "Reset".

CM_PROGRAM_PARAMETER_CHECK
Der Wert der *conversation_ID* ist ungültig.

CM_PRODUCT_SPECIFIC_ERROR
Die UPIC-Instanz konnte nicht gefunden werden.

CM_ENCRYPTION_NOT_SUPPORTED
Für diese Conversation ist keine Verschlüsselung möglich, weil entweder
- das Zusatzprodukt openUTM-Crypt nicht installiert ist.
- die UTM-Partner-Anwendung keine Verschlüsselung will, da der UPIC-Client vertrauenswürdig (trusted) ist.
- der UPIC-Client nicht verschlüsseln kann, weil das Produkt openUTM-Client ohne die Lizenz zum Verschlüsseln installiert wurde.

Zustandsänderung

Der Aufruf ändert den Zustand der Conversation nicht.

Hinweis

- CMECEL kann immer nur den aktuellen Wert der Verschlüsselungsebene liefern. Die Verschlüsselungsebene kann durch einen nachfolgenden CPI-C Aufruf immer geändert werden.
- Werden nacheinander mehrere Conversations zur gleichen Partner-Anwendung aufgebaut (d.h. die Kommunikationsverbindung wird nicht jedesmal auf- und abgebaut), so kann das Ergebnis von CMECEL nach dem ersten CMINIT CM_OK, nach allen folgenden CMINIT-Aufrufen aber CM_ENCRYPTION_NOT_SUPPORTED sein. Die UPIC-Bibliothek baut erst nach dem ersten CMALLOC-Aufruf eine Verbindung zur Partner-Anwendung auf und legt damit die Möglichkeit für Verschlüsselung fest.

Verhalten im Fehlerfall



CM_CALL_NOT_SUPPORTED
: Muss kein Fehler sein: Falls eine Anwendung sowohl für UPIC-L als auch für UPIC-R vorgesehen ist, bedeutet dieser Returncode lediglich, dass die Anwendung mit einer UPIC-L Bibliothek gebunden ist. In diesem Fall ist Verschlüsselung nicht nötig. Das Programm kann sich diesen Returncode merken und auf weitere Aufrufe zur Verschlüsselung verzichten.

CM_PROGRAM_STATE_CHECK
: Programm ändern.

CM_PROGRAM_PARAMETER_CHECK
: Programm ändern.

CM_PRODUCT_SPECIFIC_ERROR
: Das Betriebssystem kann nicht genügend Speicherplatz für interne Puffer bereitstellen. Überprüfen Sie Ihr Programm auf zu hohe Speicherplatzanforderung und starten Sie ggf. Ihr System neu.

CM_ENCRYPTION_NOT_SUPPORTED
: Muss kein Fehler sein: Falls eine UPIC-R Anwendung mit verschiedenen UTM-Partnern kommuniziert, von denen einige verschlüsseln können und andere nicht, bedeutet dieser Returncode lediglich, dass die Anwendung mit einer UTM-Anwendung kommuniziert, die nicht verschlüsseln kann oder will. In diesem Fall ist Verschlüsselung nicht möglich. Das Programm kann sich diesen Returncode merken und auf weitere Aufrufe zur Verschlüsselung verzichten.

Funktionsdeklaration: Extract_Conversation_Encryption_Level

```
Extract_Conversation_Encryption_Level (unsigned char CM_PTR conversation_ID,
                                        CM_ENCRYPTION_LEVEL   CM_PTR  encryption_level,
                                        CM_RETURN_CODE        CM_PTR  return_code )
```

Extract_Conversation_State - Zustand der Conversation abfragen

Mit dem Aufruf *Extract_Conversation_State* (CMECS) erhält ein Programm den aktuellen Zustand der Conversation.

Syntax

CMECS (conversation_ID, conversation_state, return_code)

Parameter

→ conversation_ID	Identifikation der Conversation
← conversation_state	Der Wert enthält den Zustand der Conversation. Gültige Werte für UPIC sind: – CM_INITIALIZE_STATE – CM_SEND_STATE – CM_RECEIVE_STATE
← return_code	Ergebnis des Funktionsaufrufes

Ergebnis (*return_code*)

CM_OK
: Aufruf OK

CM_PROGRAM_PARAMETER_CHECK
: Der Wert der *conversation_ID* ist ungültig.

CM_PRODUCT_SPECIFIC_ERROR
: Die UPIC Instanz konnte nicht gefunden werden.

Zustandsänderung

Der Aufruf ändert den Zustand der Conversation nicht.

Hinweis

- Falls der Returncode von CM_OK verschieden ist, hat der Wert von *conversation_state* keine Bedeutung.
- In den Zuständen "Start" und "Reset" existiert nie eine gültige *conversation_ID*.

Verhalten im Fehlerfall

CM_PROGRAM_PARAMETER_CHECK
: Programm ändern.

CM_PRODUCT_SPECIFIC_ERROR
: Das Betriebssystem kann nicht genügend Speicherplatz für interne Puffer bereitstellen. Überprüfen Sie Ihr Programm auf zu hohe Speicherplatzanforderung und starten Sie ggf. Ihr System neu.

Funktionsdeklaration: Extract_Conversation_State

```
CM_ENTRY Extract_Conversation_State (unsigned char CM_PTR conversation_ID,
                                     CM_CONVERSATION_STATE CM_PTR conversation_state,
                                              CM_RETURN_CODE CM_PTR return_code )
```

Extract_Convertion – Wert der Conversation Characteristic CHARACTER_CONVERTION abfragen

Mit dem Aufruf *Extract_Convertion* (CMECNV) erhält ein Programm den aktuellen Wert für die Characteristic *CHARACTER_CONVERTION* der Conversation.

Diese Funktion gehört zu den zusätzlichen Funktionen des Trägersystems UPIC; sie ist nicht Bestandteil der CPI-C-Schnittstelle.

Der Aufruf *Extract_Convertion* ist nur im Zustand "Init" erlaubt.

Syntax

CMECNV (conversation_ID, character_convertion, return_code)

Parameter

→ conversation_ID Identifikation der Conversation.

← character_convertion

der Wert gibt an, ob eine Code-Konvertierung der Benutzerdaten durchgeführt wird oder nicht.

Für *character_convertion* können folgende Werte zurückgegeben werden:

CM_NO_CHARACTER_CONVERTION
Es findet keine automatische Code-Konvertierung beim Senden oder Empfangen von Daten statt.

CM_IMPLICIT_CHARACTER_CONVERTION
Beim Senden und Empfangen von Daten werden die Daten automatisch konvertiert (siehe auch Abschnitt „Code-Konvertierung“ auf Seite 68).

← return_code Ergebnis des Funktionsaufrufes.

Ergebnis (*return_code*)

CM_OK
: Aufruf OK

CM_PROGRAM_PARAMETER_CHECK
: Der Wert in conversation_ID ist ungültig.

CM_PRODUCT_SPECIFIC_ERROR
: Die UPIC Instanz konnte nicht gefunden werden.

CM_PROGRAM_STATE_CHECK
: Die Conversation ist nicht im Zustand “Initialize”.

Zustandsänderung

Der Aufruf ändert den Zustand der Conversation nicht.

Hinweis

Falls der Returncode von CM_OK verschieden ist, bleibt die Characteristic *CHARACTER_CONVERTION* unverändert.

Verhalten im Fehlerfall

CM_PROGRAM_STATE_CHECK
: Programm ändern

CM_PROGRAM_PARAMETER_CHECK
: Programm ändern

CM_PRODUCT_SPECIFIC_ERROR
: Das Betriebssystem kann nicht genügend Speicherplatz für interne Puffer bereitstellen. Überprüfen Sie Ihr Programm auf zu hohe Speicherplatzanforderung und starten Sie ggf. Ihr System neu.

Funktionsdeklaration: Extract_Convertion

```
CM_ENTRY Extract_Convertion(
                       unsigned char                CM_PTR  conversation_ID,
                       CM_CHARACTER_CONVERTION_TYPE  CM_PTR  convertion_type,
                       CM_RETURN_CODE               CM_PTR  return_code )
```

Extract_Cursor_Offset - Offset der Cursor-Position abfragen

Mit dem Aufruf *Extract_Cursor_Offset* (CMECO) erhält ein Programm den zuletzt von openUTM an den Client gesendeten Offset der Cursor-Position, sofern der Cursor im UTM-Teilprogramm über KDCSCUR gesetzt wird.

Der Aufruf *Extract_Cursor_Offset* ist im Zustand "Send" und "Receive" und im Zustand "Reset" unmittelbar nach einem *Receive*-/*Receive_Mapped_Data*-Aufruf erlaubt.

Diese Funktion ist nicht Bestandteil der CPI-C Spezifikation, sondern eine zusätzliche Funktion des UPIC-Trägersystems.

Syntax

CMECO(conversation_ID, cursor_offset, return_code)

Parameter

→ conversation_ID	Identifikation der Conversation
← cursor_offset	Offset der Cursor Position
← return_code	Ergebnis des Funktionsaufrufes

Ergebnis (*return_code*)

CM_OK
: Aufruf OK

CM_CALL_NOT_SUPPORTED
: Die Funktion wird nicht unterstützt. Dieser Returncode tritt auf, wenn kein *cursor_offset* erhalten werden kann, da eine nicht mehr unterstützte Version der UTM-Partner-Anwendung eingesetzt wird.

CM_PROGRAM_PARAMETER_CHECK
: Der Wert in *conversation_ID* ist ungültig. Der Wert der *conversation_ID* ist ungültig, weil die Funktion nach Ende der Conversation mehr als einmal aufgerufen wurde oder weil noch keine Conversation existierte (nach dem *Enable_UTM_UPIC*-Aufruf ist noch kein *Initialize_Conversation*-Aufruf erfolgt).

CM_PRODUCT_SPECIFIC_ERROR
: Die UPIC-Instanz konnte nicht gefunden werden.

CM_PROGRAM_STATE_CHECK
: Die Conversation ist nicht im Zustand "Reset", "Receive" oder "Send".

Zustandsänderung

Der Aufruf ändert den Zustand der Conversation nicht.

Hinweis

- Falls der Returncode von CM_OK verschieden ist, hat der Wert von *cursor_offset* keine Bedeutung.
- Der Wert der *conversation_ID* bleibt für diesen Funktionsaufruf nach dem Ende einer Conversation so lange gültig, bis *Initialize_Conversation* oder *Extract_Cursor_Offset* aufgerufen werden.
- Ein KDCSCUR-Aufruf überschreibt einen vorhergehenden KDCSCUR-Aufruf im UTM-Teilprogramm.
- Wird im UTM-Teilprogramm bei KDCSCUR eine ungültige Adresse angegeben, liefert *Extract_Cursor_Offset* den Wert 0.
- Bei einem +Format wird für die Cursor-Position die Adresse des Attributfeldes geliefert.

Verhalten im Fehlerfall

CM_CALL_NOT_SUPPORTED
: Ist nicht unbedingt ein Fehler. Falls eine UPIC-R Anwendung mit verschiedenen UTM-Partnern kommuniziert, bedeutet dieser Returncode nur, dass die Anwendung mit einer UTM-Anwendung kommuniziert, die keinen Cursor Offset senden kann. Das Programm kann sich diesen Returncode merken und auf weitere Aufrufe für die Cursor-Position verzichten.

CM_PROGRAM_STATE_CHECK
: Programm ändern.

CM_PROGRAM_PARAMETER_CHECK
: Programm ändern.

CM_PRODUCT_SPECIFIC_ERROR
: Das Betriebssystem kann nicht genügend Speicherplatz für interne Puffer bereitstellen. Überprüfen Sie Ihr Programm auf zu hohe Speicherplatzanforderung und starten Sie ggf. Ihr System neu.

Funktionsdeklaration: Extract_Cursor_Offset

```
CM_ENTRY Extrac_Cursor_Offset ( unsigned char CM_PTR  conversation_ID,
                                     CM_INT32 CM_PTR  cursor_offset,
                               CM_RETURN_CODE CM_PTR  return_code )
```

Extract_Partner_LU_Name - partner_LU_Name abfragen

Mit dem Aufruf *Extract_Partner_LU_Name* (CMEPLN) erhält ein Programm den aktuellen *partner_LU_name* der Conversation.

Dieser Aufruf gehört zu den Advanced Functions.

Syntax

CMEPLN(conversation_ID, partner_LU_name, partner_LU_name_length, return_code)

Parameter

→ conversation_ID Identifikation der Conversation.

← partner_LU_name Gibt den *partner_LU_name* zurück. Die Länge des Parameters muss mindestens 32 Byte sein.

← partner_LU_name_length
Gibt die Länge des in *partner_LU_name* gelieferten Wertes an.
Minimum: 1, Maximum: 32.

← return_code Ergebnis des Funktionsaufrufes.

Ergebnis (*return_code*)

CM_OK
Aufruf OK

CM_PROGRAM_PARAMETER_CHECK
Der Wert in *conversation_ID* ist ungültig.

CM_PRODUCT_SPECIFIC_ERROR
Die UPIC-Instanz konnte nicht gefunden werden.

CM_PROGRAM_STATE_CHECK
Die Conversation ist nicht im Zustand „Initialize“.

Zustandsänderung

Der Aufruf ändert den Zustand der Conversation nicht.

Hinweis

- Falls der Returncode von CM_OK verschieden ist, hat der Wert von *partner_LU_name* keine Bedeutung.

Verhalten im Fehlerfall

CM_PROGRAM_PARAMETER_CHECK
: Programm ändern.

CM_PRODUCT_SPECIFIC_ERROR
: Das Betriebssystem kann nicht genügend Speicherplatz für interne Puffer bereitstellen. Überprüfen Sie Ihr Programm auf zu hohe Speicherplatzanforderung und starten Sie ggf. Ihr System neu.

CM_PROGRAM_STATE_CHECK
: Programm ändern.

Funktionsdeklaration: Extract_Partner_LU_Name

```
CM_ENTRY Extract_Partner_LU_Name (unsigned char CM_PTR conversation_ID,
                                 unsigned char CM_PTR partner_LU_name,
                                 CM_INT32 CM_PTR partner_LU_name_length,
                                 CM_RETURN_CODE CM_PTR return_code)
```

Extract_Secondary_Information - Erweiterte Information abfragen

Mit dem Aufruf *Extract_Secondary_Information* (CMESI) erhält das Programm erweiterte Informationen (secondary information), die sich auf den Returncode des letzten CPI-C-Aufrufs beziehen.

Syntax

CMESI (conversation_ID, call_ID, buffer, requested_length, data_received,
received_length, return_code)

Parameter

→ conversation_ID Identifikation der bereits initialisierten Conversation (wird vom Aufruf *Initialize* geliefert).

→ call_ID spezifiziert die Funktion, deren erweiterte Information ausgegeben werden soll.

← buffer Puffer, in dem die Daten empfangen werden. Falls der Rückgabewert von *data_received* CM_NO_DATA_RECEIVED ist, ist der Inhalt von *buffer* undefiniert.

→ requested_length Maximale Länge der Daten, die empfangen werden können.

← data_received Gibt an, ob das Programm die erweiterte Information vollständig empfangen hat. Falls das Ergebnis (*return_code*) nicht den Wert CM_OK hat, ist der Wert von *data_received* undefiniert.

data_received kann folgende Werte annehmen:
- CM_COMPLETE_DATA_RECEIVED
  Die erweiterte Information wurde vollständig empfangen.
- CM_INCOMPLETE_DATA_RECEIVED
  Die erweiterte Information ist nicht vollständig vom Programm empfangen worden.

← received_length Länge der empfangenen Daten. Der Wert von *received_length* ist undefiniert, falls das Ergebnis (*return_code*) nicht den Wert CM_OK hat.

← return_code Ergebnis des Funktionsaufrufs.

Ergebnis (*return_code*)

CM_OK
: Aufruf OK

CM_NO_SECONDARY_INFORMATION
: Für den Aufruf der angegebenen Conversation ist keine erweiterte Information vorhanden.

CM_PROGRAM_PARAMETER_CHECK
: Der Wert der *conversation_ID* ist ungültig, die *call_ID* gibt CMESI oder einen ungültigen Wert an, oder der Wert für *requested_length* ist größer als 32767 oder kleiner 1.

CM_PRODUCT_SPECIFIC_ERROR
: Die UPIC-Instanz konnte nicht gefunden werden.

Hinweis

- Das Programm sollte unmittelbar nach Erhalt eines *return_codes* diesen Aufruf machen. Nachfolgende CPI-C Aufrufe überschreiben gegebenenfalls die erweiterte Information. Wenn keine Conversation existiert, d.h. die Bibliothek ist im "Reset" Status, wird die *conversation_ID* ignoriert.
- Wenn sich der *Extract_Secondary_Information-* Aufruf erfolgreich beendet hat, wird die zurückgegebene erweiterte Information nicht länger gespeichert. Die gleiche Information ist im nachfolgenden *Extract_Secondary_Information*-Aufruf nicht mehr verfügbar.
- Das Programm kann den Aufruf nicht dazu nutzen, um von einem vorangegangenen *Extract_Secondary_Information*-Aufruf erweiterte Information zu erhalten.
- Diese Funktion wurde nicht in ihrer vollen Komplexität gemäß den CPI-C Spezifikationen implementiert. Die Vereinfachungen gegenüber CPI-C sind folgende:
  - Der interne Puffer besitzt eine beschränkte Größe von 1024 Byte.
  - Ist der Wert für *requested_length* kleiner als die Länge der intern gespeicherten erweiterten Information, wird der vom Anwendungsprogramm zur Verfügung gestellte Puffer vollständig gefüllt und *data_received* auf CM_INCOMPLETE_DATA_RECEIVED gesetzt. Es ist nicht möglich, die restlichen Daten mit weiteren CMESI-Aufrufen zu erhalten.

Verhalten im Fehlerfall

CM_PROGRAM_PARAMETER_CHECK
: Programm ändern.

CM_PRODUCT_SPECIFIC_ERROR
: Das Betriebssystem kann nicht genügend Speicherplatz für interne Puffer bereitstellen. Überprüfen Sie Ihr Programm auf zu hohe Speicherplatzanforderung und starten Sie ggf. Ihr System neu.

Funktionsdeklaration: Extract_Secondary_Information

```
CM_ENTRY Extract_Secondary_Information (
                                     unsigned char CM_PTR  conversation_ID,
                                          CM_INT32 CM_PTR  call_ID,
                                     unsigned char CM_PTR  buffer,
                                          CM_INT32 CM_PTR  requested_length,
                             CM_DATA_RECEIVED_TYPE CM_PTR  data_received,
                                          CM_INT32 CM_PTR  received_length,
                                    CM_RETURN_CODE CM_PTR  return_code )
```

Extract_Secondary_Return_Code - Erweiterten Returncode abfragen

Mit dem Aufruf *Extract_Secondary_Return_Code* (CMESRC) erhält das Programm erweiterte Returncodes (secondary return code), die sich auf den Returncode (primary return code) des letzten CPI-C-Aufrufs beziehen.

Diese Funktion gehört zu den zusätzlichen Funktionen des Trägersystems UPIC; sie ist nicht Bestandteil der CPI-C-Schnittstelle.

Syntax

CMESRC (conversation_ID, call_ID, secondary_return_code, return_code)

Parameter

→ conversation_ID
: Identifikation der bereits initialisierten Conversation (wird vom Aufruf *Initialize* geliefert).

→ call_ID
: Spezifiziert die Funktion, deren erweiterter Returncode ausgegeben werden soll.

← secondary_return_code
: Gibt den erweiterten Returncode des letzten CPI-C-Aufrufs zurück. Falls das Ergebnis ungleich CM_OK ist, ist der Wert für *secondary_return_code* undefiniert.

← return_code
: Ergebnis des Funktionsaufrufs.

Ergebnis (*return_code*)

CM_OK
: Aufruf OK

CM_NO_SECONDARY_RETURN_CODE
: Für den Aufruf der angegebenen Conversation ist kein erweiterter Returncode vorhanden.

CM_PROGRAM_PARAMETER_CHECK
: Der Wert der *conversation_ID* ist ungültig, die *call_ID* gibt CMESRC oder einen ungültigen Wert an.

CM_PRODUCT_SPECIFIC_ERROR
: Die UPIC-Instanz konnte nicht gefunden werden.

Erweiterter Returncode (*secondary_return_code*)

CM_SECURITY_USER_UNKNOWN
Die angegebene Benutzerkennung ist nicht generiert.

CM_SECURITY_STA_OFF
Die angegebene Benutzerkennung ist durch Generierung oder Administration gesperrt.

Der Administrator der UTM-Anwendung kann die Sperre aufheben.

CM_SECURITY_USER_IS_WORKING
Mit dieser Benutzerkennung hat sich bereits jemand an dieser UTM-Anwendung angemeldet.

CM_SECURITY_OLD_PSWORD_WRONG
Das angegebene bisherige Passwort ist falsch.

CM_SECURITY_NEW_PSWORD_WRONG
Die Angaben zum neuen Passwort sind nicht verwendbar. Mögliche Ursache: minimale Gültigkeitsdauer noch nicht abgelaufen.

Altes Passwort bis zum Ablauf der Gültigkeitsdauer weiterverwenden.

CM_SECURITY_NO_CARD_READER
Der Benutzer ist mit Magnetstreifenkarte generiert und kann sich nicht über UPIC anmelden.

CM_SECURITY_CARD_INFO_WRONG
Der Benutzer ist mit Chipkarte generiert und kann sich nicht über UPIC anmelden.

CM_SECURITY_NO_RESOURCES
Die Anmeldung ist zur Zeit nicht möglich. Ursache ist
- ein Betriebsmittelengpass oder
- die Maximalzahl gleichzeitig angemeldeter Benutzer ist erreicht (siehe KDCDEF-Anweisung MAX CONN-USERS=) oder
- ein inverser KDCDEF läuft gerade

Anmeldung später wieder versuchen.

CM_SECURITY_NO_KERBEROS_SUPPORT
Der Benutzer ist mit einem Kerberos-Prinzipal generiert und kann sich nicht über UPIC anmelden.

CM_SECURITY_TAC_KEY_MISSING
Das aktuelle LTERM hat nicht die Berechtigung, den Vorgang fortzusetzen.

CM_SECURITY_PWD_EXPIRED_NO_RETRY
Die Gültigkeitsdauer des Benutzer-Passwortes ist abgelaufen, die UTM-Anwendung ist mit SIGNON GRACE=NO generiert.

Der Client-Anwender kann sich nicht mehr anmelden. Er muss den Administrator der UTM-Anwendung darum bitten, ein neues Passwort für ihn einzutragen.

CM_SECURITY_COMPLEXITY_ERROR
Das neue Passwort erfüllt nicht die Anforderung an die Komplexität. Siehe KDCDEF Steueranweisung USER PROTECT-PW= .

CM_SECURITY_PASSWORD_TOO_SHORT
Das neue Passwort erfüllt nicht die Anforderung an die Mindestlänge.
Siehe KDCDEF Steueranweisung USER PROTECT-PW=.

CM_SECURITY_UPD_PSWORD_WRONG
Das von KDCUPD übertragene Passwort erfüllt nicht die in der Anwendungsgenerierung definierte Komplexitätsstufe oder Mindestlänge.
Siehe KDCDEF Steueranweisung USER PROTECT-PW= .

Das Passwort muss per Administration geändert werden, bevor der Benutzer sich wieder anmelden kann.

CM_SECURITY_TA_RECOVERY
Für die angegebene Benutzerkennung ist ein Transaktionswiederanlauf erforderlich.

CM_SECURITY_PROTOCOL_CHANGED
Der Benutzer hat einen offenen Vorgang, der nicht von einem UPIC-Client aus fortgesetzt werden kann.

CM_SECURITY_SHUT_WARN
Der Anwendungslauf wird beendet, nur Benutzer mit Administrationsberechtigung dürfen sich noch anmelden.

Die Anmeldung ist erst wieder möglich, wenn die UTM-Anwendung neu gestartet worden ist.

CM_SECURITY_ENC_LEVEL_TOO_HIGH
Auf der Verbindung ist der für die Fortsetzung des offenen Vorgangs nötige Verschlüsselungsmechanismus nicht verfügbar.

CM_SECURITY_PWD_EXPIRED_RETRY
Die Gültigkeitsdauer des Benutzer-Passworts ist abgelaufen, die UTM-Anwendung ist mit SIGNON GRACE=YES generiert.

Der Client kann sich trotzdem anmelden, wenn er beim Anmelden zusätzlich zu seinem bisherigen Passwort ein geeignetes neues Passwort angibt.

Wenn das neue Passwort gleich dem bisherigen ist, dann lehnt openUTM die Anmeldung ab. Bei Zusammenarbeit mit openUTM > 5.1A30 setzt UPIC in diesem Fall als erweiterten Returncode CM_SECURITY_NEW_PSWORD_WRONG.

Die folgenden sekundären Returncodes treten nur im Zusammenhang mit UTM-Cluster-Anwendungen auf:

CM_SECURITY_USER_GLOBALLY_UNKNOWN
Die angegebene Benutzerkennung ist in der Cluster-User-Datei nicht bekannt.

CM_SECURITY_USER_SIGNED_ON_OTHER_NODE
Mit dieser Benutzerkennung hat sich bereits ein Benutzer an einer anderen Knoten-Anwendung angemeldet.

CM_SECURITY_TRANSIENT_ERROR
Beim Anmelden trat ein temporärer Fehler auf. Auf die Cluster-User-Datei konnte innerhalb der in der Knoten-Anwendung konfigurierten Zeit nicht zugegriffen werden.

Anmeldung später noch einmal versuchen.

Hinweis

- Das Programm sollte diesen Aufruf unmittelbar nach Erhalt eines Returncodes machen. Nachfolgende CPI-C Aufrufe überschreiben gegebenenfalls den erweiterten Returncode. Wenn keine Conversation existiert, d.h. die Bibliothek ist im Status "Reset", wird die *conversation_ID* ignoriert.
- Wenn sich der *Extract_Secondary_Return_Code*-Aufruf erfolgreich beendet hat, wird der zurückgegebene erweiterte Returncode nicht länger gespeichert. Der gleiche Returncode ist im nachfolgenden *Extract_Secondary_Return_Code*-Aufruf nicht mehr verfügbar.
- Das Programm kann den Aufruf nicht dazu nutzen, um von einem vorangegangenen *Extract_Secondary_Return_Code*-Aufruf einen erweiterten Returncode zu erhalten.
- Den erweiterten Returncode und die Beschreibung finden Sie bei den einzelnen UPIC-Aufrufen.

Zustandsänderung

Keine Zustandsänderung.

Verhalten im Fehlerfall

CM_PROGRAM_PARAMETER_CHECK
: Programm ändern.

CM_PRODUCT_SPECIFIC_ERROR
: Das Betriebssystem kann nicht genügend Speicherplatz für interne Puffer bereitstellen. Überprüfen Sie Ihr Programm auf zu hohe Speicherplatzanforderung und starten Sie ggf. Ihr System neu.

Funktionsdeklaration: Extract_Secondary_Return_Code

```
CM_ENTRY Extract_Secondary_Return_Code (
                        unsigned char CM_PTR  conversation_ID,
                        CM_INT32      CM_PTR  call_ID,
                        CM_RETURN_CODE CM_PTR  secondary_return_code,
                        CM_RETURN_CODE CM_PTR  return_code )
```

Extract_Shutdown_State - Shutdown-Status des Servers abfragen

Mit dem Aufruf *Extract_Shutdown_State* (CMESHS) erhält ein Programm den aktuellen Shutdown-Status der UTM-Partner-Anwendung.

Der Aufruf *Extract_Shutdown_State* ist im Zustand "Send" und "Receive" und im Zustand "Reset" unmittelbar nach einem *Receive-/Receive_Mapped_Data*-Aufruf erlaubt.

Diese Funktion ist nicht Bestandteil der CPI-C Spezifikation, sondern eine zusätzliche Funktion des UPIC-Trägersystems.

Syntax

CMESHS (conversation_ID, shutdown_state, return_code)

Parameter

→ conversation_ID Identifikation der Conversation

← shutdown_state Der Wert enthält den Shutdown-Status der UTM-Partner-Anwendung. Gültige Werte sind:

- CM_SHUTDOWN_NONE:
  Die Anwendung hat keinen Shutdown eingeleitet.
- CM_SHUTDOWN_WARN:
  Die Anwendung hat SHUTDOWN WARN eingeleitet.
- CM_SHUTDOWN_GRACE:
  Die Anwendung hat SHUTDOWN GRACE eingeleitet.

← return_code Ergebnis des Funktionsaufrufs

Ergebnis (*return_code*)

CM_OK
: Aufruf OK

CM_CALL_NOT_SUPPORTED
: Die Funktion wird nicht unterstützt. Dieser Returncode tritt auf, wenn kein Shutdown-Status erhalten werden kann, da die UTM-Partner-Anwendung mit Version < V6.1 dies nicht unterstützt.

CM_PROGRAM_PARAMETER_CHECK
: Der Wert in *conversation_ID* ist ungültig.
Der Wert der *conversation_ID* ist ungültig, weil die Funktion nach Ende der Conversation mehr als einmal aufgerufen wurde oder weil noch keine Conversation existierte (nach dem *Enable_UTM_UPIC*-Aufruf ist noch kein *Initialize_Conversation*-Aufruf erfolgt).

CM_PRODUCT_SPECIFIC_ERROR
Die UPIC-Instanz konnte nicht gefunden werden.

Zustandsänderung

Der Aufruf ändert den Zustand der Conversation nicht.

Hinweis

- Falls der Returncode von CM_OK verschieden ist, hat der Wert von *shutdown_state* keine Bedeutung.
- Der Wert der *conversation_ID* bleibt für diesen Funktionsaufruf nach dem Ende einer Conversation so lange gültig, bis *Initialize_Conversation* oder *Extract_Shutdown_State* aufgerufen werden.

Verhalten im Fehlerfall

CM_CALL_NOT_SUPPORTED
Ist nicht unbedingt ein Fehler des Programms. Falls eine UPIC-R-Anwendung mit verschiedenen UTM-Partnern kommuniziert, bedeutet dieser Returncode lediglich, dass die Anwendung mit einer UTM-Partner-Anwendung kommuniziert, die keinen Shutdown-Status senden kann (openUTM < V6.1). Das Programm kann sich diesen Returncode merken und auf weitere *Extract_Shutdown_State*-Aufrufe verzichten.

CM_PROGRAM_PARAMETER_CHECK
Programm ändern.

CM_PRODUCT_SPECIFIC_ERROR
Das Betriebssystem kann nicht genügend Speicherplatz für interne Puffer bereitstellen. Überprüfen Sie Ihr Programm auf zu hohe Speicherplatzanforderung und starten Sie ggf. Ihr System neu.

Funktionsdeklaration: Extract_Shutdown_State

```
CM_ENTRY Extract_Shutdown_State(
                        unsigned char      CM_PTR  conversation_ID,
                        CM_SHUTDOWN_STATE  CM_PTR  shutdown_sate,
                        CM_RETURN_CODE     CM_PTR  return_code )
```

Extract_Shutdown_Time - Shutdown-Time des Servers abfragen

Mit dem Aufruf *Extract_Shutdown_Time* (CMESHT) erhält ein Programm die aktuelle Shutdown-Time der UTM-Partner-Anwendung.

Die Shutdown-Time, die zurückgeliefert wird, wird abdruckbar in der Länge *received_length* geliefert und hat das Zeitformat Universal Time Coordinated (UTC). Sie muss noch in die lokale Zeitzone umgerechnet werden.

Der Aufruf *Extract_Shutdown_Time* ist im Zustand "Send" und "Receive" und im Zustand "Reset" unmittelbar nach einem *Receive-/Receive_Mapped_Data*-Aufruf sowie nach einem *Extract_Shutdown_State*-Aufruf erlaubt.

Diese Funktion ist nicht Bestandteil der CPI-C Spezifikation, sondern eine zusätzliche Funktion des UPIC-Trägersystems.

Syntax

CMESHT (conversation_ID, buffer, requested_length, data_received, received_length, return_code)

Parameter

→ conversation_ID Identifikation der Conversation

← buffer Puffer, in dem die Daten empfangen werden. Falls der Rückgabewert von *data_received* CM_NO_DATA_RECEIVED ist, ist der Inhalt von *buffer* undefiniert.

In *buffer* wird der Zeitpunkt zurückgeliefert, zu dem die Anwendung heruntergefahren wird. Die einzelnen Bytes haben folgene Bedeutung:

Byte 1 - 8: Datum im Format jjjjmmtt:

jjjj	Jahr, vierstellig
mm	Monat
tt	Tag

Byte 9 - 11

ttt	Tag im Jahr

Byte 12 - 17: Uhrzeit im Format hhmmss (UTC-Format):

hh	Stunde
mm	Minute
ss	Sekunde

→ requested_length Maximale Länge der Daten, die empfangen werden können.

← data_received Gibt an, ob das Programm die Daten vollständig empfangen hat.

Falls das Ergebnis (*return_code*) nicht einen der Werte CM_OK oder CM_DEALLOCATED_NORMAL hat, ist der Wert von *data_received* undefiniert.

data_received kann folgende Werte annehmen:

CM_COMPLETE_DATA_RECEIVED
Die Daten wurden vollständig empfangen.

CM_INCOMPLETE_DATA_RECEIVED
Die Daten wurden nicht vollständig empfangen.

CM_NO_DATA_RECEIVED
Es wurden keine Daten empfangen.

← received_length Länge der empfangenen Daten. Der Wert von *received_length* ist undefiniert, wenn das Ergebnis (*return_code*) ungleich CM_OK ist.

← return_code Ergebnis des Funktionsaufrufs

Ergebnis (*return_code*)

CM_OK
Aufruf OK

CM_CALL_NOT_SUPPORTED
Die Funktion wird nicht unterstützt. Dieser Returncode tritt auf, wenn keine Shutdown-Time erhalten werden kann, da die UTM-Partner-Anwendung mit Version < 6.1 dies nicht unterstützt.

CM_PROGRAM_PARAMETER_CHECK
Der Wert in *conversation_ID* ist ungültig.
Der Wert der *conversation_ID* ist ungültig, weil die Funktion nach Ende der Conversation mehr als einmal aufgerufen wurde oder weil noch keine Conversation existierte (nach dem *Enable_UTM_UPIC*-Aufruf ist noch kein *Initialize_Conversation*-Aufruf erfolgt).
Oder der Wert für *requested_length* ist größer als 32767 oder kleiner als 1.

CM_PRODUCT_SPECIFIC_ERROR
Die UPIC-Instanz konnte nicht gefunden werden.

Hinweis

- Diese Funktion wurde nicht in ihrer vollen Komplexität gemäß den CPI-C Spezifikationen implementiert. Die Vereinfachungen gegenüber CPI-C sind folgende:
  - Der interne Puffer besitzt eine beschränkte Größe von 1024 Byte.
  - Ist der Wert für *requested_length* kleiner als die Länge der intern gespeicherten erweiterten Information, wird der vom Anwendungsprogramm zur Verfügung gestellte Puffer vollständig gefüllt und *data_received* auf CM_INCOMPLETE_DATA_RECEIVED gesetzt. Es ist nicht möglich, die restlichen Daten mit weiteren CMESHT-Aufrufen zu erhalten.
- Der Wert der *conversation_ID* bleibt für diesen Funktionsaufruf nach dem Ende einer Conversation so lange gültig, bis *Initialize_Conversation* oder *Extract_Shutdown_Time* aufgerufen werden.

Verhalten im Fehlerfall

CM_CALL_NOT_SUPPORTED
: Ist nicht unbedingt ein Fehler des Programms. Falls eine UPIC-R-Anwendung mit verschiedenen UTM-Partnern kommuniziert, bedeutet dieser Returncode lediglich, dass die Anwendung mit einer UTM-Partner-Anwendung kommuniziert, die keine Shutdown-Time senden kann (openUTM < V6.1). Das Programm kann sich diesen Returncode merken und auf weitere *Extract_Shutdown_Time*-Aufrufe verzichten.

CM_PROGRAM_PARAMETER_CHECK
: Programm ändern.

CM_PRODUCT_SPECIFIC_ERROR
: Das Betriebssystem kann nicht genügend Speicherplatz für interne Puffer bereitstellen. Überprüfen Sie Ihr Programm auf zu hohe Speicherplatzanforderung und starten Sie ggf. Ihr System neu.

Funktionsdeklaration: Extract_Shutdown_Time

```
CM_ENTRY Extract_Shutdown_Time(
                      unsigned char      CM_PTR  conversation_ID,
                      unsigned char      CM_PTR  buffer,
                           CM_INT32      CM_PTR  requested_length,
                CM_DATA_RECEIVED_TYPE    CM_PTR  data_received,
                           CM_INT32      CM_PTR  received_length,
                     CM_RETURN_CODE      CM_PTR  return_code )
```

Extract_Transaction_State - Vorgangs- und Transaktionsstatus des Servers abfragen

Mit dem Aufruf *Extract_Transaction_State* erhält ein Programm den von openUTM an den Client gesendeten Vorgangs- und Transaktionsstatus.

Der Aufruf *Extract_Transaction_State* ist nur im Zustand "Send" und "Receive" und im Zustand "Reset" unmittelbar nach einem *Receive*-/*Receive_Mapped_Data*-Aufruf erlaubt.

Diese Funktion ist nicht Bestandteil der CPI-C Spezifikation, sondern eine zusätzliche Funktion des UPIC-Trägersystems.

Syntax

CMETS (conversation_ID, transaction_state, requested_length, transaction_state_length, return_code)

Parameter

→ conversation_ID Identifikation der Conversation

← transaction_state Transaktions- und Vorgangs-Status

→ requested_length Maximale Länge der Daten, die empfangen werden können

← transaction_state_length
Länge der empfangenen Nachricht

← return_code Ergebnis des Funktionsaufrufs

Ergebnis (*return_code*)

CM_OK
Aufruf OK

CM_CALL_NOT_SUPPORTED
Die Funktion wird nicht unterstützt. Dieser Returncode tritt auf, wenn kein *transaction_state* erhalten werden kann.

CM_PROGRAM_PARAMETER_CHECK
Der Wert in *conversation_ID* ist ungültig.
Der Wert der *conversation_ID* ist ungültig, wenn die Funktion nach Ende der Conversation mehr als einmal aufgerufen wurde oder wenn noch keine Conversation existierte (nach dem *Enable_UTM_UPIC*-Aufruf ist noch kein *Initialize_Conversation*-Aufruf erfolgt).

CM_PRODUCT_SPECIFIC_ERROR
Die UPIC-Instanz konnte nicht gefunden werden.

CM_PROGRAM_STATE_CHECK
Die Conversation ist nicht im Zustand "Reset", "Send" oder "Receive"

Zustandsänderung

Der Aufruf ändert den Zustand der Conversation nicht.

Hinweis

- Falls der Returncode von CM_OK verschieden ist, hat der Wert von *transaction_state* keine Bedeutung.
- Der Wert der *conversation_ID* bleibt für diesen Funktionsaufruf nach dem Ende einer Conversation so lange gültig, bis ein *Initialize_Conversation*- oder ein *Extract_Transaction_State*-Aufruf erfolgt ist.
- Wenn der Wert von *transaction_state_length* gleich 0 ist, dann wurde kein neuer *transaction_state* empfangen.

Verhalten im Fehlerfall

CM_CALL_NOT_SUPPORTED
Ist nicht unbedingt ein Fehler des Programms. Falls eine UPIC-R-Anwendung mit verschiedenen UTM-Partnern kommuniziert, bedeutet dieser Returncode lediglich, dass die Anwendung mit einer UTM-Partner-Anwendung kommuniziert, die keinen Transaktions- und Vorgangs-Status senden kann. Das Programm kann sich diesen Returncode merken und auf weitere *Extract_Transaction_State*-Aufrufe verzichten.

CM_PROGRAM_STATE_CHECK
Programm ändern.

CM_PROGRAM_PARAMETER_CHECK
Programm ändern.

CM_PRODUCT_SPECIFIC_ERROR
Das Betriebssystem kann nicht genügend Speicherplatz für interne Puffer bereitstellen. Überprüfen Sie Ihr Programm auf zu hohe Speicherplatzanforderung und starten Sie ggf. Ihr System neu.

Beschreibung *transaction_state*

Die ersten beiden Byte des *transaction_state* enthalten die Information über den Vorgangs- und Transaktionsstatus des Servers und können entsprechend ausgewertet werden, die restlichen Byte (dd dd) enthalten interne Diagnoseinformationen.

transaction_state (hexadezimal)	**Bedeutung**
17 08 dd dd 18 08 dd dd	Ende des Verarbeitungsschritts; die Transaktion ist nicht abgeschlossen, der Vorgang ist noch offen (PEND/PGWT KP).
15 06 dd dd 16 06 dd dd	Ende des Verarbeitungsschritts; die Transaktion ist abgeschlossen, der Vorgang ist noch offen (PEND RE/PGWT CM).
1A 04 dd dd	Ende eines Vorgangs und Ende der Transaktion (PEND FI).
30 04 dd dd	Ende eines Vorgangs mit Speicherabzug (PEND ER).
31 04 dd dd	Ende eines Vorgangs (System PEND ER, d.h. PEND ER durch openUTM).
32 04 dd dd	Ende eines Vorgangs wegen abnormaler Taskbeendigung (nur openUTM auf BS2000/OSD)
20 04 dd dd 21 04 dd dd	Rücksetzen der ersten Transaktion eines Vorgangs und Vorgang beenden (PEND RS).
20 06 dd dd 21 06 dd dd	Rücksetzen einer Folgetransaktion auf den letzten Sicherungspunkt; der Vorgang ist noch offen (PEND RS).

Näheres zum PEND- und PGWT-Aufruf siehe openUTM-Handbuch „Anwendungen programmieren mit KDCS“.

Funktionsdeklaration: Extract_Transaction_State

```
CM_ENTRY Extract_Transaction_State(
                        unsigned char     CM_PTR  conversation_ID,
                        unsigned char     CM_PTR  transaction_state,
                        CM_INT32          CM_PTR  requested_length,
                        CM_INT32          CM_PTR  transaction_state_length,
                        CM_RETURN_CODE    CM_PTR  return_code )
```

Initialize_Conversation - Conversation Characteristics initialisieren

Der Aufruf *Initialize_Conversation* (CMINIT) liest den durch den *symbolic destination name* spezifizierten Eintrag in der `upicfile` und initialisiert die Conversation Characteristics. Die Characteristics *partner_LU_name*, *partner_LU_name_lth*, *TP_name* und *TP_name_length* werden mit den entsprechenden Werten aus der `upicfile` besetzt. Alle anderen Conversation Characteristics werden mit den Standardwerten initialisiert.

Neben den Conversation Characteristics wird bei diesem Aufruf auch festgelegt, ob bei den nachfolgenden *Send*- bzw. *Receive*-Aufrufen eine automatische Konvertierung der Benutzerdaten von ASCII nach EBCDIC bzw. umgekehrt stattfinden soll. Die Konvertierung erfolgt:



- in Unix- und Windows-Systemen, falls vor dem *symbolic destination name* das Kennzeichen HD steht



- In BS2000/OSD, falls vor dem *symbolic destination name* das Kennzeichen SD steht.

Näheres siehe auch Abschnitt „Side Information für stand-alone UTM-Anwendungen“ auf Seite 298.

Als Ergebnis liefert die Funktion eine achtstellige Conversation_ID zurück. Diese dient als eindeutige Identifikation der Conversation und muss in allen späteren CPI-C-Aufrufen verwendet werden, um die Conversation zu adressieren.

Es besteht die Möglichkeit, zu einem späteren Zeitpunkt die mit diesem Aufruf initialisierten Werte für die Conversation Characteristics *TP_name*, *TP_name_length, receive_type* und *deallocate_type* zu ändern. Dazu stehen die Aufrufe *Set_TP_Name*, *Set_Receive_Type* und *Set_Deallocate_Type* zur Verfügung. Ein mit einem Set-Aufruf geänderter Wert bleibt bis zum Ende der Conversation oder bis zu einem erneuten Set-Aufruf bestehen.

Die Set-Aufrufe sind kein Bestandteil des CPI-C-Starter-Sets, sondern gehören zu den Advanced Functions.

Syntax

CMINIT (conversation_ID, sym_dest_name, return_code)

Parameter

← conversation_ID Identifikation, die der Conversation zugeordnet wurde und dem Programm als Ergebnisparameter zurückgeliefert wird.

→ sym_dest_name Falls Sie ohne `upicfile` arbeiten, dann müssen Sie für *sym_dest_name* 8 Leerzeichen angeben („leerer *sym_dest_name*“).

Falls Sie mit der `upicfile` arbeiten, geben Sie den Verweis auf die Side Information ein (8 Zeichen langer Name). Für *sym_dest_name* können Sie auch 8 Leerzeichen angeben („leerer *sym_dest_name*“). In diesem Fall wird in der Side Information der symbolic destination name .DEFAULT gesucht (siehe Abschnitt „Side Information für stand-alone UTM-Anwendungen“ auf Seite 298) und die entsprechenden Werte für *partner_LU_name*, *partner_LU_name_lth*, *TP_name* und *TP_name_length* gesetzt.

← return_code Ergebnis des Funktionsaufrufs

Ergebnis (*return_code*)

CM_OK

Aufruf ok

CM_PROGRAM_PARAMETER_CHECK

- Der Wert für *sym_dest_name* bzw. *local_name* (beim *Enable_UTM_UPIC*) ist ungültig oder der spezifizierte Eintrag in der `upicfile` konnte nicht gelesen werden oder ist syntaktisch ungültig.
- Ein eventuelles An- oder Abmelden von der Transportschnittstelle war nicht erfolgreich.
- In *sym_dest_name* oder in *local_name* (beim *Enable_UTM_UPIC*) wurde ein leerer Name angegeben, aber in der `upicfile` fehlt ein entsprechender DEFAULT-Eintrag bzw. der DEFAULT-Eintrag ist ungültig.
- Fehler in der `upicfile`:
  Die CD-Einträge für den angegebenen *sym_dest_name* folgen nicht unmittelbar aufeinander oder die CD-Einträge für den angegebenen *sym_dest_name* beinhalten unterschiedliche TACs.

CM_PRODUCT_SPECIFIC_ERROR

- Für dieses Programm ist bereits eine Conversation aktiv, bzw. es wurde noch kein *Enable_UTM_UPIC*-Aufruf gemacht.
- Eine unerwartete Reaktion der Transportschnittstelle ist aufgetreten.

Zustandsänderung

Falls das Ergebnis CM_OK ist, geht das Programm in den Zustand "Initialize" über und die Characteristics der Conversation sind initialisiert. Näheres siehe Abschnitt „Conversation Characteristics“ auf Seite 51. In allen Fehlersituationen ändert das Programm seinen Zustand nicht.

Hinweis

- Der *Initialize_Conversation*-Aufruf muss vom Programm ausgeführt worden sein, bevor ein anderer Aufruf für diese Conversation erfolgt.
- Falls das Programm mit dem *Initialize_Conversation*-Aufruf oder daran anschließenden Set-Aufrufen ungültige Information für das Etablieren einer Conversation bereitstellt, wird dies bei syntaktischen Fehlern sofort, bei inhaltlichen Fehlern jedoch erst bei der Ausführung des *Allocate*-Aufrufs (CMALLC) erkannt.
- Mehrere Programme können sich unter dem gleichen Namen anmelden, wenn für die entsprechende TPOOL-Anweisung CONNECT-MODE=MULTI definiert ist.
- bei remote Anbindung:
  - Die Funktion führt eventuell ein Anmelden an das Transportsystem (z.B. TCP/IP, PCMX, CMX, BCAM) durch. Dazu wird der Name des vorangegangenen *Enable_UTM_UPIC*-Aufrufs verwendet. Falls das Programm bereits mit demselben Namen angemeldet ist, erfolgt kein Anmelden.
  - Besteht noch eine Verbindung zu einem Partner, der ungleich dem Partner aus der `upicfile` ist, dann wird diese Verbindung abgebaut.

X/W
- bei lokaler Anbindung (UPIC auf Unix- und Windows-Systemen):

X/W X/W X/W X/W X/W X/W X/W
  - Die Funktion führt die Anmeldung an die openUTM-interne Prozesskommunikation durch (mit dem UTM-Anwendungsnamen aus der `upicfile`), wenn das Programm noch nicht mit demselben Namen angemeldet ist. Ist das Programm noch mit einem anderen Namen angemeldet, erfolgt zuerst eine Abmeldung von der openUTM-internen Prozesskommunikation. Eine bestehende Conversation zu dieser UTM-Anwendung wird dabei implizit abgebaut. Erst danach wird das Programm mit dem neuen Namen angemeldet.

X/W X/W X/W
  - Bei der Anmeldung an die UTM-Anwendung wird die *applifile* der UTM-Anwendung gelesen. Dazu wird die Shellvariable UTMPATH, die auf das entsprechende UTM-Verzeichnis *utmpfad* zeigt, ausgewertet. Diese Variable muss gesetzt sein.

Verhalten im Fehlerfall

CM_PROGRAM_PARAMETER_CHECK

- `upicfile` einrichten oder die Umgebungsvariable bzw. Jobvariable UPICPATH richtig setzen. Überprüfen des TNS-Eintrags bzw. des BCMAP-Eintrags in BS2000/OSD.
- Den aktuellen *sym_dest_name* in die `upicfile` eintragen oder den Eintrag für *sym_dest_name* auf richtige Syntax prüfen.



- bei lokaler Anbindung: Umgebungsvariable UTMPATH richtig setzen. Es ist auch möglich, dass kein Semaphor mehr zur Verfügung steht.
- `upicfile` ändern: CD-Einträge überprüfen und anpassen.

CM_PRODUCT_SPECIFIC_ERROR

Programm ändern oder Systemdienst informieren und Diagnoseunterlagen erstellen.

Funktionsdeklaration: Initialize_Conversation

```
CM_ENTRY Initialize_Conversation ( unsigned char CM_PTR  conversation_ID,
                                   unsigned char CM_PTR  sym_dest_name,
                                  CM_RETURN_CODE CM_PTR  return_code)
```

Prepare_To_Receive - Vom Sende- in den Empfangsstatus wechseln

Der Aufruf *Prepare_To_Receive* (CMPTR) bewirkt folgendes:

– Alle Daten, die zum Zeitpunkt des Aufrufs noch im lokalen Sendepuffer gespeichert sind, werden zusammen mit dem Senderecht an den UTM-Vorgang übertragen.

– Nachdem die Daten aus dem Sendepuffer an den UTM-Vorgang übergeben sind, geht die Conversation vom Zustand "Send" in den Zustand "Receive" über.

Prepare_To_Receive darf nur aufgerufen werden, wenn sich die Conversation im Zustand "Send" befindet, jedoch nicht direkt nach dem *Allocate*-Aufruf bzw. nach dem Empfang des Senderechts vom Partner. In diesen beiden Ausnahmefällen muss ein *Send_Data*- oder *Send_Mapped_Data*-Aufruf vor dem *Prepare_To_Receive*-Aufruf abgesetzt werden.

Nach dem *Prepare_To_Receive*-Aufruf muss als nächstes *Receive* bzw. *Receive_Mapped_Data* aufgerufen werden. Vor dem *Receive-* bzw. *Receive_Mapped_Data*-Aufruf darf jedoch *Set_Receive_Timer* oder *Set_Receive_Type* aufgerufen werden.

Syntax

CMPTR (conversation_ID, return_code)

Parameter

→ conversation_ID	Identifikation der Conversation
← return_code	Ergebnis des Funktionsaufrufs

Ergebnis (*return_code*)

CM_OK

Aufruf ok. Die Conversation ist von "Send" in den Zustand "Receive" übergegangen.

CM_DEALLOCATED_ABEND

mögliche Ursachen:

- Abnormale Beendigung des UTM-Vorgangs
- UTM-Anwendungsende
- Verbindungsabbau durch die UTM-Administration
- Verbindungsabbau durch das Transportsystem
- Verbindungsabbau durch openUTM wegen Überschreitung der maximal zulässigen Anzahl von gleichzeitig angemeldeten Benutzern (MAX-Anweisung, CONN-USERS=). Dieser Fall kann auch dann auftreten, wenn beim Aufruf *Set_Conversation_Security_User_ID* eine Administrator-Benutzerkennung übergeben wurde. Das ist dann der Fall, wenn dem LTERM-Partner des CPI-C-Programms in der UTM-Anwendung eine Benutzerkennung zugeordnet ist (über LTERM ...USER=), die keine Administrationsberechtigung hat.

Das Programm geht in den Zustand "Reset" über.

CM_PRODUCT_SPECIFIC_ERROR

mögliche Ursachen:

- Die UPIC-Instanz konnte nicht gefunden werden.
- Der *Prepare_To_Receive*-Aufruf erfolgte unmittelbar nach einem *Allocate*-Aufruf anstatt eines *Send_Data*- bzw. *Send_Mapped_Data*-Aufrufs.

CM_PROGRAM_STATE_CHECK

Der Aufruf ist im aktuellen Zustand der Conversation nicht erlaubt.

CM_PROGRAM_PARAMETER_CHECK

Der Wert in *conversation_ID* ist ungültig.

CM_RESOURCE_FAILURE_NO_RETRY

Es ist ein Fehler aufgetreten, der zu einer vorzeitigen Beendigung der Conversation führte (z.B. ein Protokollfehler oder vorzeitiger Verlust der Netzverbindung). Das Programm geht in den Zustand "Reset" über.

Zustandsänderung

- Ist das Ergebnis des Aufrufs CM_OK, dann ändert sich der Zustand der Conversation von "Send" nach "Receive".
- Bei folgenden Ergebnissen geht das Programm in den Zustand "Reset" über:
  CM_DEALLOCATED_ABEND
  CM_RESOURCE_FAILURE_NO_RETRY
- Bei allen anderen Fehlersituationen ändert das Programm seinen Zustand nicht.

Verhalten im Fehlerfall

CM_PRODUCT_SPECIFIC_ERROR
- Programm ändern.
- Das Betriebssystem kann nicht genügend Speicherplatz für interne Puffer bereitstellen. Überprüfen Sie Ihr Programm auf zu hohe Speicherplatzanforderung und starten Sie ggf. Ihr System neu.

CM_PROGRAM_STATE_CHECK
Programm ändern.

CM_PROGRAM_PARAMETER_CHECK
Programm ändern.

CM_RESOURCE_FAILURE_NO_RETRY
Systemdienst informieren und Diagnoseunterlagen erstellen. Es kann auch eine Störung im Transportsystem die Ursache für diesen Fehlercode sein.

Funktionsdeklaration: Prepare_To_Receive

```
CM_ENTRY Prepare_To_Receive (unsigned char CM_PTR  conversation_ID,
                             CM_RETURN_CODE CM_PTR  return_code )
```

Receive - Daten von einem UTM-Service empfangen

Mit dem Aufruf *Receive* (CMRCV) empfängt ein Programm Informationen von einem UTM-Service.

Der Aufruf kann blockierend oder nicht-blockierend ausgeführt werden.

– Der *Receive*-Aufruf ist blockierend, wenn die Characteristic *receive_type* den Wert CM_RECEIVE_AND_WAIT hat.
  Liegen zum Zeitpunkt des *Receive-*Aufrufs keine Informationen (Daten oder Senderecht) vor, dann wartet der Programmlauf so lange im *Receive*, bis eine Information für diese Conversation vorliegt. Erst dann kehrt der Programmlauf aus dem *Receive*-Aufruf zurück und liefert die Informationen zurück. Falls zum Zeitpunkt des Aufrufs bereits eine Information vorliegt, empfängt sie das Programm, ohne zu warten.

  Um die Wartezeit beim blockierenden *Receive*-Aufruf zu beschränken, sollten entsprechende Timer in der UTM-Partner-Anwendung gesetzt werden.

– Der *Receive*-Aufruf ist nicht-blockierend, wenn die Characteristic *receive_type* den Wert CM_RECEIVE_IMMEDIATE hat.
  Liegen zum Zeitpunkt des *Receive*-Aufrufs keine Informationen vor, dann wartet der Programmlauf nicht, bis Informationen für diese Conversation eintreffen. Der Programmlauf kehrt sofort aus dem *Receive*-Aufruf zurück. Falls bereits eine Information vorliegt, wird sie an das Programm übergeben.



  UPIC-Local:
  Der nicht-blockierende *Receive*-Aufruf wird bei der Anbindung über UPIC-Local nicht unterstützt.

Die Characteristic *receive_type* können Sie vor dem Aufruf von *Receive* mit dem Aufruf *Set_Receive_Type* setzen. Nach dem Initialisieren einer Conversation ist standardmäßig der blockierende Receive eingestellt.

Syntax

CMRCV (conversation_ID, buffer, requested_length, data_received, received_length,
status_received, control_information_received, return_code)

Parameter

→ conversation_ID
: Identifikation der Conversation

← buffer
: Puffer, in dem die Daten empfangen werden. Falls der Rückgabewert von *data_received* CM_NO_DATA_RECEIVED ist, ist der Inhalt von *buffer* undefiniert.

→ requested_length
: Maximale Länge der Daten, die empfangen werden können.

← data_received
: Gibt an, ob das Programm Daten empfangen hat.

Falls das Ergebnis (*return_code*) nicht einen der Werte CM_OK oder CM_DEALLOCATED_NORMAL hat, ist der Wert von *data_received* undefiniert.

data_received kann folgende Werte annehmen:

CM_NO_DATA_RECEIVED
Es lagen keine Daten für das Programm vor. Eventuell wurde das Senderecht empfangen.

CM_COMPLETE_DATA_RECEIVED
Eine Nachricht, die für das Programm vorlag, wurde vollständig empfangen.

CM_INCOMPLETE_DATA_RECEIVED
Eine Nachricht ist nicht vollständig an das Programm übergeben worden. Falls *data_received* diesen Wert annimmt, muss das Programm anschließend so viele *Receive*-Aufrufe absetzen, bis die Nachricht vollständig übergeben wurde, d.h. bis *data_received* den Wert CM_COMPLETE_DATA_RECEIVED hat.

← received_length
: Länge der empfangenen Daten. Der Wert von *received_length* ist undefiniert, wenn das Programm keine Daten empfangen hat (*data_received*=CM_NO_DATA_RECEIVED) bzw. wenn das Ergebnis ungleich CM_OK oder CM_DEALLOCATE_NORMAL ist.

← status_received
: Gibt an, ob das Programm das Senderecht empfangen hat.

status_received kann einen der folgenden Werte annehmen:

CM_NO_STATUS_RECEIVED
Das Senderecht wurde nicht empfangen.

CM_SEND_RECEIVED
Der UTM-Vorgang hat das Senderecht an das Programm abgegeben. Das Programm muss anschließend einen *Send_Data*-Aufruf absetzen.

Falls das Ergebnis ungleich CM_OK ist, ist der Wert für *status_received* undefiniert.

← control_information_received
Wird nur syntaktisch unterstützt und kann nur den Wert CM_REQ_TO_SEND_NOT_RECEIVED annehmen.

Der Wert in *control_information_received* ist undefiniert, wenn das Ergebnis in *return_code* ungleich CM_OK oder CM_DEALLOCATE_NORMAL ist.

← return_code Ergebnis des Funktionsaufrufs

Ergebnis (*return_code*)

CM_OK

Falls das Ergebnis CM_OK ist, hat das Programm nach dem Aufruf einen der folgenden Zustände:

"Receive" falls *status_received* den Wert CM_NO_STATUS_RECEIVED hat.
"Send" falls der Wert von *status_received* CM_SEND_RECEIVED ist.

CM_SECURITY_NOT_VALID

mögliche Ursachen:

- ungültige UTM-Benutzerkennung bei *Set_Conversation_Security_User_ID*
- ungültiges Passwort beim Aufruf *Set_Conversation_Security_Password*
- Die UTM-Anwendung ist ohne USER generiert
- Der User kann sich bei der UTM-Anwendung wegen Betriebsmittelengpass nicht anmelden

Wenn die UPIC-Anwendung mit einer UTM-Anwendung kommuniziert, die das Ergebnis der Berechtigungsprüfung detailliert zurückliefert, dann liefert die UPIC-Bibliothek einen erweiterten Returncode, der die Ursache detailliert beschreibt. Die Ergebnisse, die das Programm dann erhält, sind unter *secondary_return_code* aufgeführt, siehe Seite 158.

Die erweiterten Returncodes können auch durch den Aufruf *Extract_Secondary_Return_Code* abgefragt werden, siehe Seite 131.

CM_TPN_NOT_RECOGNIZED

mögliche Ursachen:

- ungültiger Transaktionscode (TAC) in der `upicfile` oder beim *Set_TP_Name*-Aufruf, z.B.:
  - TAC ist nicht generiert
  - Keine Berechtigung, um diesen TAC aufzurufen
  - TAC ist nur als Folge-TAC erlaubt
  - TAC ist kein Dialog-TAC
  - TAC ist mit Verschlüsselung generiert, aber es wurden unverschlüsselte Benutzerdaten gesendet oder auf der Verbindung wird keine Verschlüsselung unterstützt oder die verschlüsselten Daten entsprechen nicht der geforderten Verschlüsselungsstufe.
- Vorgangs-Wiederanlauf mit Hilfe von KDCDISP wurde abgewiesen, da keine mit RESTART=YES generierte UTM-Benutzerkennung angegeben wurde.

CM_TP_NOT_AVAILABLE_NO_RETRY

Vorgangs-Wiederanlauf mit Hilfe von KDCDISP ist nicht möglich, da UTM-Anwendung neu generiert wurde.

CM_TP_NOT_AVAILABLE_RETRY

Vorgangsstart wurde abgewiesen, da UTM-Anwendung beendet wird.

CM_DEALLOCATED_ABEND

mögliche Ursachen:

- Abnormale Beendigung des UTM-Vorgangs
- UTM-Anwendungsende
- Verbindungsabbau durch UTM-Administration
- Verbindungsabbau durch das Transportsystem
- Verbindungsabbau durch openUTM wegen Überschreitung der maximal zulässigen Anzahl von Benutzern (MAX-Anweisung, CONN-USERS=). Die Ursache kann auch darin liegen, dass beim Aufruf *Set_Conversation_Security_User_ID* zwar eine Administrator-Benutzerkennung übergeben wurde, aber die per UTM-Generierung der Verbindung implizit zugeordnete Benutzerkennung oder die explizit (mit der Anweisung LTERM..., USER=) zugeordnete (Verbindungs-)Benutzerkennung keine Administrator-Benutzerkennung ist (CONN-USERS wirkt nur für Benutzer ohne Administrationsberechtigung).

Das Programm geht in den Zustand "Reset" über.

CM_DEALLOCATED_NORMAL

Im UTM-Vorgang wurde ein PEND-FI-Aufruf ausgeführt. Das Programm geht in den Zustand "Reset" über.

CM_RESOURCE_FAILURE_RETRY
: Ein vorübergehender Betriebsmittelengpass führte zur Beendigung der Conversation. Möglicherweise können im UTM-Pagepool keine Daten mehr zwischengespeichert werden. Tritt der Fehler häufiger auf, sollte der Pagepool der UTM-Anwendung vergrößert werden (MAX-Anweisung, PGPOOL=).

CM_RESOURCE_FAILURE_NO_RETRY
: Es ist ein Fehler aufgetreten, der zu einer vorzeitigen Beendigung der Conversation führte (z.B. ein Protokollfehler oder vorzeitiger Verlust der Netzverbindung).

CM_PROGRAM_STATE_CHECK
: Der Aufruf ist im aktuellen Zustand nicht erlaubt. Der Inhalt aller anderen Variablen ist undefiniert.

CM_PROGRAM_PARAMETER_CHECK
: Der Wert in *conversation_ID* ist ungültig, oder der Wert in *requested_length* ist größer als 32767 oder kleiner als Null. Der Inhalt aller anderen Variablen ist undefiniert.

CM_PRODUCT_SPECIFIC_ERROR
: Anstatt eines *Send_Data*-Aufrufs erfolgte ein *Receive*-Aufruf (nur unmittelbar nach einem *Allocate*-Aufruf).

CM_OPERATION_INCOMPLETE
: Der Aufruf *Receive* ist durch den Ablauf des Timers, der mit *Set_Receive_Timer* gesetzt wurde, unterbrochen worden. Es wurden keine Daten empfangen.

CM_UNSUCCESSFUL
: *receive_type* hat den Wert CM_RECEIVE_IMMEDIATE und es sind zur Zeit keine Daten für die Conversation vorhanden.

Erweiterter Returncode (*secondary_return_code*)

CM_SECURITY_USER_UNKNOWN
: Die angegebene Benutzerkennung ist nicht generiert.

CM_SECURITY_STA_OFF
: Die angegebene Benutzerkennung ist gesperrt.

CM_SECURITY_USER_IS_WORKING
: Mit dieser Benutzerkennung hat sich bereits jemand angemeldet.

CM_SECURITY_OLD_PSWORD_WRONG
: Das angegebene bisherige Passwort ist falsch.

CM_SECURITY_NEW_PSWORD_WRONG
: Die Angaben zum neuen Passwort sind nicht verwendbar. Mögliche Ursache: minimale Gültigkeitsdauer noch nicht abgelaufen.

CM_SECURITY_NO_CARD_READER
: Der Benutzer ist mit Magnetstreifenkarte generiert und kann sich nicht über UPIC anmelden.

CM_SECURITY_CARD_INFO_WRONG
: Der Benutzer ist mit Chipkarte generiert und kann sich nicht über UPIC anmelden.

CM_SECURITY_NO_RESOURCES
: Die Anmeldung ist zur Zeit nicht möglich. Ursache ist
  - ein Betriebsmittelengpass oder
  - die Maximalzahl gleichzeitig angemeldeter Benutzer ist erreicht (siehe KDCDEF-Anweisung MAX CONN-USERS=) oder
  - ein inverser KDCDEF läuft gerade

  Anmeldung später wieder versuchen.

CM_SECURITY_NO_KERBEROS_SUPPORT
: Der Benutzer ist mit einem Kerberos-Prinzipal generiert und kann sich nicht über UPIC anmelden.

CM_SECURITY_TAC_KEY_MISSING
: Das aktuelle LTERM hat nicht die Berechtigung, den Vorgang fortzusetzen.

CM_SECURITY_PWD_EXPIRED_NO_RETRY
: Die Gültigkeitsdauer des Benutzer-Passwortes ist abgelaufen.

CM_SECURITY_COMPLEXITY_ERROR
: Das neue Passwort erfüllt nicht die Anforderung an die Komplexität.

CM_SECURITY_PASSWORD_TOO_SHORT
: Das neue Passwort ist zu kurz.

CM_SECURITY_UPD_PSWORD_WRONG
: Das von KDCUPD übertragene Passwort erfüllt nicht die in der Anwendungs-generierung definierte Komplexitätsstufe.

CM_SECURITY_TA_RECOVERY
: Für die angegebene Benutzerkennung ist ein Transaktionswiederanlauf erforder-lich.

CM_SECURITY_PROTOCOL_CHANGED
: Der offene Vorgang kann nicht von diesem LTERM-Partner aus fortgesetzt werden.

CM_SECURITY_SHUT_WARN
: Vom Administrator wurde SHUT WARN gegeben, normale Benutzer dürfen sich nicht mehr an die UTM-Anwendung anmelden, nur ein Administrator darf sich noch anmelden.

CM_SECURITY_ENC_LEVEL_TOO_HIGH
: Auf der Verbindung ist der für die Fortsetzung des offenen Vorgangs nötige Verschlüsselungsmechanismus nicht verfügbar.

CM_SECURITY_PWD_EXPIRED_RETRY
: Die Gültigkeitsdauer des Benutzer-Passworts ist abgelaufen.

Die folgenden sekundären Returncodes treten nur im Zusammenhang mit UTM-Cluster-Anwendungen auf:

CM_SECURITY_USER_GLOBALLY_UNKNOWN
: Die angegebene Benutzerkennung ist in der Cluster-User-Datei nicht bekannt.

CM_SECURITY_USER_SIGNED_ON_OTHER_NODE
: Mit dieser Benutzerkennung hat sich bereits ein Benutzer an einer anderen Knoten-Anwendung angemeldet.

CM_SECURITY_TRANSIENT_ERROR
: Beim Anmelden trat ein temporärer Fehler auf. Auf die Cluster-User-Datei konnte innerhalb der in der Knoten-Anwendung konfigurierten Zeit nicht zugegriffen werden.

: Anmeldung später noch einmal versuchen.

Zustandsänderung

– Falls das Ergebnis CM_OK ist, hat das Programm nach dem Aufruf einen der folgenden Zustände:

  "Receive" falls *status_received* den Wert CM_NO_STATUS_RECEIVED hat.
  "Send" falls *status_received* den Wert CM_SEND_RECEIVED hat.

– Das Programm geht bei folgenden Ergebnissen in den Zustand "Reset" über:
  CM_DEALLOCATED_ABEND
  CM_DEALLOCATED_NORMAL
  CM_SECURITY_NOT_VALID
  CM_TPN_NOT_RECOGNIZED
  CM_TP_NOT_AVAILABLE_RETRY/NO_RETRY
  CM_RESOURCE_FAILURE_RETRY/NO_RETRY
  CM_SECURITY_USER_UNKNOWN
  CM_SECURITY_STA_OFF
  CM_SECURITY_USER_IS_WORKING
  CM_SECURITY_OLD_PSWORD_WRONG
  CM_SECURITY_NEW_PSWORD_WRONG
  CM_SECURITY_NO_CARD_READER
  CM_SECURITY_CARD_INFO_WRONG
  CM_SECURITY_NO_RESOURCES
  CM_SECURITY_NO_KERBEROS_SUPPORT
  CM_SECURITY_TAC_KEY_MISSING
  CM_SECURITY_PWD_EXPIRED_NO_RETRY
  CM_SECURITY_COMPLEXITY_ERROR
  CM_SECURITY_PASSWORD_TOO_SHORT
  CM_SECURITY_UPD_PSWORD_WRONG
  CM_SECURITY_TA_RECOVERY
  CM_SECURITY_PROTOCOL_CHANGED
  CM_SECURITY_SHUT_WARN
  CM_SECURITY_ENC_LEVEL_TOO_HIGH
  CM_SECURITY_PWD_EXPIRED_RETRY
  CM_SECURITY_USER_GLOBALLY_UNKNOWN
  CM_SECURITY_USER_SIGNED_ON_OTHER_NODE
  CM_SECURITY_TRANSIENT_ERROR

– Bei allen anderen Fehlersituationen ändert das Programm seinen Zustand nicht.

Hinweis

- Wurde vor einem blockierenden *Receive*-Aufruf mit dem Aufruf *Set_Receive_Timer* eine maximale Wartezeit eingestellt, dann kehrt der Programmlauf spätestens nach Ablauf der Wartezeit aus dem *Receive*-Aufruf zurück und der *Receive*-Aufruf liefert dann das Ergebnis (*return_code)* CM_OPERATION_INCOMPLETE zurück.
- Bei einem *Receive*-Aufruf kann ein Programm nur so viele Daten empfangen, wie im Parameter *requested_length* angegeben wurde. Deshalb ist es möglich, dass eine Nachricht mit dem *Receive*-Aufruf nur teilweise gelesen wird. Ob eine Nachricht komplett gelesen wurde oder nicht, können Sie dem Wert des Parameters *data_received* entnehmen:
  - Falls das Programm bereits die komplette Nachricht empfangen hat, hat der Parameter *data_received* den Wert CM_COMPLETE_DATA_RECEIVED.
  - Hat das Programm noch nicht alle Daten der Nachricht empfangen, dann hat der Parameter *data_received* den Wert CM_INCOMPLETE_DATA_RECEIVED. Das Programm muss dann solange *Receive* aufrufen, bis *data_received* den Wert CM_COMPLETE_DATA_RECEIVED hat.
- Mit einem einzigen Aufruf kann ein Programm sowohl Daten als auch das Senderecht empfangen. Die Parameter *return_code*, *data_received* und *status_received* geben Auskunft über die Art der Information, die ein Programm erhalten hat.
- Falls das Programm den *Receive*-Aufruf im Zustand "Send" absetzt, wird das Senderecht an den UTM-Vorgang abgegeben. Auf diese Weise wird die Senderichtung der Conversation geändert.
- Ein *Receive*-Aufruf mit *requested_length*=0 hat keine spezielle Bedeutung.
  Falls Daten vorliegen, werden diese in der Länge 0 empfangen mit *data_received*=CM_INCOMPLETE_DATA_RECEIVED.
  Falls keine Daten vorliegen, kann das Senderecht empfangen werden. D.h. es können entweder Daten oder das Senderecht empfangen werden, aber nicht beides.
- Übergibt die UTM-Partner-Anwendung ein Formatkennzeichen (Strukturierungsmerkmale der übergebenen Daten), dann wird dieses zwar von UPIC empfangen (im UTM-Vorgang tritt kein Fehler auf), kann aber nicht an das Programm übergeben werden. Daten zusammen mit Formatkennzeichen können nur mit *Receive_Mapped_Data* gelesen werden.

Verhalten im Fehlerfall

CM_RESOURCE_FAILURE_RETRY
Conversation neu einrichten.

CM_RESOURCE_FAILURE_NO_RETRY
Systemdienst informieren und Diagnoseunterlagen erstellen. Es kann auch eine Störung im Transportsystem die Ursache für diesen Fehlercode sein.

CM_PROGRAM_STATE_CHECK
Programm ändern.

CM_PROGRAM_PARAMETER_CHECK
Programm ändern.

CM_PRODUCT_SPECIFIC_ERROR
Programm ändern.

CM_SECURITY_USER_UNKNOWN
Die UTM-Benutzerkennung ist nicht generiert. Benutzerkennung verwenden, die generiert ist oder gewünschte Benutzerkennung generieren oder dynamisch konfigurieren.

CM_SECURITY_STA_OFF
Benutzerkennung mit STATUS=ON generieren oder per Administration entsperren.

CM_SECURITY_USER_IS_WORKING
Andere UTM-Benutzerkennung benutzen oder den Vorgang des bereits angemeldeten Benutzers beenden.

CM_SECURITY_OLD_PSWORD_WRONG
Passwort korrekt angegeben.

CM_SECURITY_NEW_PSWORD_WRONG
Altes Passwort bis Ablauf der Gültigkeitsdauer weiterverwenden.

CM_SECURITY_NO_CARD_READER
Der Benutzer ist mit Magnetstreifenkarte generiert und kann sich nicht über UPIC anmelden.

CM_SECURITY_CARD_INFO_WRONG
Der Benutzer ist mit Chipkarte generiert.

CM_SECURITY_NO_RESOURCES
Später wieder probieren.

CM_SECURITY_NO_KERBEROS_SUPPORT
Der Benutzer ist mit einem Kerberos-Prinzipal generiert und kann sich nicht über UPIC anmelden.

CM_SECURITY_TAC_KEY_MISSING
: Generierung oder Programm ändern.

CM_SECURITY_PWD_EXPIRED_NO_RETRY
: Die Gültigkeitsdauer des Passworts ist abgelaufen. Das Passwort muss per Administration geändert werden, bevor der Benutzer sich wieder anmelden kann.

CM_SECURITY_COMPLEXITY_ERROR
: Das neue Passwort entsprechend den Anforderungen der generierten Komplexitätsstufe wählen, siehe KDCDEF-Anweisung USER PROTECT-PW=.

CM_SECURITY_PASSWORD_TOO_SHORT
: Neues längeres Passwort verwenden oder Generierung ändern, siehe KDCDEF-Anweisung USER PROTECT-PW= *length*, ... (Wert für die minimale Länge).

CM_SECURITY_UPD_PSWORD_WRONG
: Das Passwort entspricht nicht der geforderten Komplexitätsstufe oder hat nicht die erforderliche Länge, siehe KDCDEF-Anweisung USER PROTECT-PW=. Das Passwort muss per Administration geändert werden, bevor sich der Benutzer wieder anmelden kann.

CM_SECURITY_TA_RECOVERY
: Für die angegebene Benutzerkennung ist ein Transaktionswiederanlauf nötig.

CM_SECURITY_PROTOCOL_CHANGED
: Der Benutzer hat einen offenen Vorgang, der nicht von einem UPIC-Client aus fortgesetzt werden kann.

CM_SECURITY_SHUT_WARN
: Die UTM-Anwendung wird beendet; es dürfen sich nur noch Benutzer mit Administrationsberechtigung anmelden. Abwarten, bis die Anwendung neu gestartet wurde.

CM_SECURITY_ENC_LEVEL_TOO_HIGH
: Auf der Verbindung ist der für die Fortsetzung des offenen Vorgangs nötige Verschlüsselungsmechanismus nicht verfügbar.

CM_SECURITY_PWD_EXPIRED_RETRY
: Den Aufbau der Conversation mit Angabe des alten und eines neuen Passworts wiederholen.

Die folgenden sekundären Returncodes treten nur im Zusammenhang mit UTM-Cluster-Anwendungen auf:

CM_SECURITY_USER_GLOBALLY_UNKNOWN
: Die angegebene Benutzerkennung ist in der Cluster-User-Datei nicht bekannt.

CM_SECURITY_USER_SIGNED_ON_OTHER_NODE
: Mit dieser Benutzerkennung hat sich bereits ein Benutzer an einer anderen Knoten-Anwendung angemeldet.

CM_SECURITY_TRANSIENT_ERROR
: Beim Anmelden trat ein temporärer Fehler auf. Auf die Cluster-User-Datei konnte innerhalb der in der Knoten-Anwendung konfigurierten Zeit nicht zugegriffen werden.

  Anmeldung später noch einmal versuchen.

Funktionsdeklaration: Receive

```
CM_ENTRY Receive ( unsigned char CM_PTR  conversation_ID,
                   unsigned char CM_PTR  buffer,
                        CM_INT32 CM_PTR  requested_length,
           CM_DATA_RECEIVED_TYPE CM_PTR  data_received,
                        CM_INT32 CM_PTR  received_length,
              CM_STATUS_RECEIVED CM_PTR  status_received,
 CM_CONTROL_INFORMATION_RECEIVED CM_PTR  control_information_received,
                  CM_RETURN_CODE CM_PTR  return_code )
```

Receive_Mapped_Data - Daten und Formatkennzeichen von einem UTM-Service empfangen

Mit dem *Receive_Mapped_Data* (CMRCVM)-Aufruf empfängt ein Programm Informationen von einem UTM-Service. Die Informationen, die empfangen werden, können entweder Daten, ein Formatkennzeichen und/oder das Senderecht sein.

Der Aufruf kann blockierend oder nicht-blockierend ausgeführt werden.

– Der *Receive_Mapped_Data*-Aufruf ist blockierend, wenn die Characteristic *receive_type* den Wert CM_RECEIVE_AND_WAIT hat.
  Liegen zum Zeitpunkt des *Receive_Mapped_Data-*Aufrufs keine Informationen (Daten oder Senderecht) vor, dann wartet der Programmlauf so lange im *Receive_Mapped_Data*, bis eine Information für diese Conversation eintrifft. Erst dann kehrt der Programmlauf aus dem *Receive_Mapped_Data*-Aufruf zurück und liefert die Informationen zurück. Falls zum Zeitpunkt des Aufrufs bereits eine Information vorliegt, empfängt sie das Programm ohne zu warten.

  Um die Wartezeit beim blockierenden *Receive_Mapped_Data*-Aufruf zu beschränken, sollten entsprechende Timer in der UTM-Partner-Anwendung gesetzt werden.

– Der *Receive_Mapped_Data*-Aufruf ist nicht-blockierend, wenn die Characteristic *receive_type* den Wert CM_RECEIVE_IMMEDIATE hat.
  Liegen zum Zeitpunkt des *Receive_Mapped_Data*-Aufrufs keine Informationen vor, dann wartet der Programmlauf <u>nicht,</u> bis Informationen für diese Conversation eintreffen. Der Programmlauf kehrt sofort aus dem *Receive_Mapped_Data*-Aufruf zurück. Falls bereits eine Information vorliegt, wird sie an das Programm übergeben.

Die Characteristic *receive_type* können Sie vor dem Aufruf von *Receive_Mapped_Data* mit dem Aufruf *Set_Receive_Type* setzen.

Syntax

CMRCVM (conversation_ID, map_name, map_name_length, buffer, requested_length,
data_received, received_length, status_received,
control_information_received, return_code)

Parameter

→ conversation_ID	Identifikation der Conversation
← map_name	Formatkennzeichen, das die UTM-Partner-Anwendung zusammen mit den Daten an das CPI-C-Programm gesendet hat. Das Formatkennzeichen spezifiziert die Strukturierungsmerkmale der empfangenen Daten.
← map_name_length	Länge des Formatkennzeichens in *map_name*.

← buffer
: Puffer, in dem die Daten empfangen werden. Falls der Rückgabewert von *data_received* CM_NO_DATA_RECEIVED ist, ist der Inhalt von *buffer* undefiniert.

→ requested_length
: Maximale Länge der Daten, die empfangen werden können.

← data_received
: Gibt an, ob auf der Conversation Daten empfangen wurden.

data_received kann folgende Werte annehmen:

CM_NO_DATA_RECEIVED
Es lagen keine Daten für das Programm vor. Eventuell wurde jedoch das Senderecht empfangen.

CM_COMPLETE_DATA_RECEIVED
Eine Nachricht, die für das Programm vorlag, wurde vollständig empfangen.

CM_INCOMPLETE_DATA_RECEIVED
Eine Nachricht ist nicht vollständig an das Programm übergeben worden. Falls *data_received* diesen Wert annimmt, muss das Programm anschließend so viele *Receive*- bzw. *Receive_Mapped_Data*-Aufrufe absetzen, bis die Nachricht vollständig gelesen wurde, d.h. bis *data_received* den Wert CM_COMPLETE_DATA_RECEIVED hat.
Der Wert von *data_received* ist undefiniert, wenn das Ergebnis des Aufrufs ungleich CM_OK oder CM_DEALLOCATED_NORMAL ist.

← received_length
: Länge der empfangenen Daten. Der Wert von *received_length* ist undefiniert, wenn das Programm keine Daten empfangen hat (*data_received*=CM_NO_DATA_RECEIVED) bzw. wenn das Ergebnis ungleich CM_OK oder CM_DEALLOCATE_NORMAL ist.

← status_received
: Gibt an, ob das Programm das Senderecht empfangen hat.

status_received kann einen der folgenden Werte annehmen:

CM_NO_STATUS_RECEIVED
Das Senderecht wurde nicht empfangen.

CM_SEND_RECEIVED
Der UTM-Vorgang hat das Senderecht an das Programm abgegeben. Das Programm muss anschließend einen *Send_Data*-Aufruf absetzen.

Falls das Ergebnis ungleich CM_OK ist, ist der Wert für *status_received* undefiniert.

← control_information_received
Wird nur syntaktisch unterstützt und kann nur den Wert CM_REQ_TO_SEND_NOT_RECEIVED annehmen.

Der Wert in *control_information_received* ist undefiniert, wenn das Ergebnis in *return_code* ungleich CM_OK oder CM_DEALLOCATE_NORMAL ist.

← return_code Ergebnis des Funktionsaufrufs

Ergebnis (*return_code*)

CM_OK

Aufruf ok. Das Programm hat nach dem Aufruf einen der folgenden Zustände:

"Receive" falls *status_received* den Wert CM_NO_STATUS_RECEIVED hat.
"Send" falls *status_received* den Wert CM_SEND_RECEIVED hat.

CM_SECURITY_NOT_VALID

mögliche Ursachen:
- ungültige UTM-Benutzerkennung bei *Set_Conversation_Security_User_ID*
- ungültiges Passwort beim Aufruf *Set_Conversation_Security_Password*
- Die UTM-Anwendung ist ohne Benutzerkennungen (USER-Anweisungen) generiert.
- Der Benutzer kann sich bei der UTM-Anwendung wegen Betriebsmittelengpass nicht anmelden.

Wenn die UPIC-Anwendung mit einer UTM-Anwendung kommuniziert, die das Ergebnis der Berechtigungsprüfung detailliert zurückliefert, dann liefert die UPIC-Bibliothek einen erweiterten Returncode, der die Ursache detailliert beschreibt. Die Ergebnisse, die das Programm dann erhält, sind unter *secondary_return_code* aufgeführt, siehe Seite 170.

Die erweiterten Returncodes können auch durch den Aufruf *Extract_Secondary_Return_Code* abgefragt werden, siehe Seite 131.

CM_TPN_NOT_RECOGNIZED

mögliche Ursachen:

- Vorgangs-Wiederanlauf mit Hilfe von KDCDISP wurde abgewiesen, da keine mit RESTART=YES generierte UTM-Benutzerkennung angegeben wurde.
- ungültiger Transaktionscode (TAC) in der `upicfile` oder beim *Set_TP_Name*-Aufruf, z.B.:
  - TAC ist nicht generiert
  - Keine Berechtigung, um diesen TAC aufzurufen
  - TAC ist nur als Folge-TAC erlaubt
  - TAC ist kein Dialog-TAC
  - TAC ist mit Verschlüsselung generiert, aber es wurden unverschlüsselte Benutzerdaten gesendet oder auf der Verbindung wird keine Verschlüsselung unterstützt oder die verschlüsselten Daten entsprechen nicht der geforderten Verschlüsselungsstufe.
- Vorgangs-Wiederanlauf mit Hilfe von KDCDISP wurde abgewiesen, da keine mit RESTART=YES generierte UTM-Benutzerkennung angegeben wurde.

CM_TP_NOT_AVAILABLE_NO_RETRY

Vorgangs-Wiederanlauf mit Hilfe von KDCDISP ist nicht möglich, da UTM-Anwendung neu generiert wurde.

CM_TP_NOT_AVAILABLE_RETRY

Vorgangsstart wurde abgewiesen, da UTM-Anwendung beendet wird.

CM_DEALLOCATED_ABEND

mögliche Ursachen:

- Abnormale Beendigung des UTM-Vorgangs
- UTM-Anwendungsende
- Verbindungsabbau durch UTM-Administration
- Verbindungsabbau durch das Transportsystem
- Verbindungsabbau durch openUTM wegen Überschreitung der maximal zulässigen Anzahl von Benutzern (MAX-Anweisung, CONN-USERS=). Die Ursache kann auch darin liegen, dass beim Aufruf *Set_Conversation_Security_User_ID* zwar eine Administrator-Benutzerkennung übergeben wurde, aber die per UTM-Generierung der Verbindung implizit zugeordnete Benutzerkennung oder die explizit (mit der Anweisung LTERM..., USER=) zugeordnete (Verbindungs-)Benutzerkennung keine Administrator-Benutzerkennung ist (CONN-USERS wirkt nur für Benutzer ohne Administrationsberechtigung).

Das Programm geht in den Zustand "Reset" über.

CM_DEALLOCATED_NORMAL

Im UTM-Vorgang wurde ein PEND-FI-Aufruf ausgeführt. Das Programm geht in den Zustand "Reset" über.

CM_OPERATION_INCOMPLETE
: Der Aufruf *Receive_Mapped_Data* ist durch den Ablauf des Timers, der mit *Set_Receive_Timer* gesetzt wurde, unterbrochen worden. Es wurden keine Daten empfangen.

CM_UNSUCCESSFUL
: Die Characteristic *receive_type* hat den Wert CM_RECEIVE_IMMEDIATE und es sind zur Zeit keine Daten für die Conversation vorhanden.

CM_RESOURCE_FAILURE_RETRY
: Ein vorübergehender Betriebsmittelengpass führte zur Beendigung der Conversation. Möglicherweise können im UTM-Pagepool keine Daten mehr zwischengespeichert werden.

  Maßnahme: UTM-Pagepool vergrößern (MAX-Anweisung, PGPOOL=).

CM_RESOURCE_FAILURE_NO_RETRY
: Es ist ein Fehler aufgetreten, der zu einer vorzeitigen Beendigung der Conversation führte (z.B. ein Protokollfehler oder vorzeitiger Verlust der Netzverbindung).

CM_PROGRAM_STATE_CHECK
: Der Aufruf ist im aktuellen Zustand nicht erlaubt. Der Inhalt aller anderen Variablen ist undefiniert.

CM_PROGRAM_PARAMETER_CHECK
: Der Wert in *conversation_ID* ist ungültig, oder der Wert in *requested_length* ist größer als 32767 oder kleiner als 0. Der Inhalt aller anderen Variablen ist undefiniert.

CM_PRODUCT_SPECIFIC_ERROR
: Anstatt eines *Send_Data*-Aufrufs erfolgte ein *Receive*-Aufruf (nur unmittelbar nach einem *Allocate*-Aufruf).

CM_MAP_ROUTINE_ERROR
: In der UTM-Partner-Anwendung werden keine Formatkennzeichen im UPIC-Protokoll unterstützt.

Erweiterter Returncode (*secondary_return_code*)

CM_SECURITY_USER_UNKNOWN
: Die angegebene Benutzerkennung ist nicht generiert.

CM_SECURITY_STA_OFF
: Die angegebene Benutzerkennung ist gesperrt.

CM_SECURITY_USER_IS_WORKING
: Mit dieser Benutzerkennung hat sich bereits jemand angemeldet.

CM_SECURITY_OLD_PSWORD_WRONG
Das angegebene bisherige Passwort ist falsch.

CM_SECURITY_NEW_PSWORD_WRONG
Die Angaben zum neuen Passwort sind nicht verwendbar. Mögliche Ursache: minimale Gültigkeitsdauer noch nicht abgelaufen.

CM_SECURITY_NO_CARD_READER
Der Benutzer ist mit Magnetstreifenkarte generiert und kann sich nicht über UPIC anmelden.

CM_SECURITY_CARD_INFO_WRONG
Der Benutzer ist mit Chipkarte generiert und kann sich nicht über UPIC anmelden.

CM_SECURITY_NO_RESOURCES
Die Anmeldung ist zur Zeit nicht möglich. Ursache ist

- ein Betriebsmittelengpass oder
- die Maximalzahl gleichzeitig angemeldeter Benutzer ist erreicht (siehe KDCDEF-Anweisung MAX CONN-USERS=) oder
- ein inverser KDCDEF läuft gerade

Anmeldung später wieder versuchen.

CM_SECURITY_NO_KERBEROS_SUPPORT
Der Benutzer ist mit einem Kerberos-Prinzipal generiert und kann sich nicht über UPIC anmelden.

CM_SECURITY_TAC_KEY_MISSING
Das aktuelle LTERM hat nicht die Berechtigung, den Vorgang fortzusetzen.

CM_SECURITY_PWD_EXPIRED_NO_RETRY
Die Gültigkeitsdauer des Benutzer-Passwortes ist abgelaufen.

CM_SECURITY_COMPLEXITY_ERROR
Das neue Passwort erfüllt nicht die Anforderung an die Komplexität.

CM_SECURITY_PASSWORD_TOO_SHORT
Das neue Passwort ist zu kurz.

CM_SECURITY_UPD_PSWORD_WRONG
Das von KDCUPD übertragene Passwort erfüllt nicht die in der Anwendungs-generierung definierte Komplexitätsstufe.

CM_SECURITY_TA_RECOVERY
Für die angegebene Benutzerkennung ist ein Transaktionswiederanlauf erforder-lich.

CM_SECURITY_PROTOCOL_CHANGED
Der offene Vorgang kann nicht von diesem LTERM-Partner aus fortgesetzt werden.

CM_SECURITY_SHUT_WARN
: Vom Administrator wurde SHUT WARN gegeben, normale Benutzer dürfen sich nicht mehr an die UTM-Anwendung anmelden, nur ein Administrator darf sich noch anmelden.

CM_SECURITY_ENC_LEVEL_TOO_HIGH
: Auf der Verbindung ist der für die Fortsetzung des offenen Vorgangs nötige Verschlüsselungsmechanismus nicht verfügbar.

CM_SECURITY_PWD_EXPIRED_RETRY
: Die Gültigkeitsdauer des Benutzer-Passworts ist abgelaufen.

Die folgenden sekundären Returncodes treten nur im Zusammenhang mit UTM-Cluster-Anwendungen auf:

CM_SECURITY_USER_GLOBALLY_UNKNOWN
: Die angegebene Benutzerkennung ist in der Cluster-User-Datei nicht bekannt.

CM_SECURITY_USER_SIGNED_ON_OTHER_NODE
: Mit dieser Benutzerkennung hat sich bereits ein Benutzer an einer anderen Knoten-Anwendung angemeldet.

CM_SECURITY_TRANSIENT_ERROR
: Beim Anmelden trat ein temporärer Fehler auf. Auf die Cluster-User-Datei konnte innerhalb der in der Knoten-Anwendung konfigurierten Zeit nicht zugegriffen werden.

: Anmeldung später noch einmal versuchen.

Zustandsänderung

– Falls das Ergebnis CM_OK ist, hat das Programm nach dem Aufruf einen der folgenden Zustände:

  "Receive" falls der Wert von *status_received* CM_NO_STATUS_RECEIVED ist.
  "Send" falls der Wert von *status_received* CM_SEND_RECEIVED ist.

– Das Programm geht bei folgenden Ergebnissen in den Zustand "Reset" über:
  CM_DEALLOCATED_ABEND
  CM_DEALLOCATED_NORMAL
  CM_SECURITY_NOT_VALID
  CM_TPN_NOT_RECOGNIZED
  CM_TP_NOT_AVAILABLE_RETRY/NO_RETRY
  CM_RESOURCE_FAILURE_RETRY/NO_RETRY
  CM_SECURITY_USER_UNKNOWN
  CM_SECURITY_STA_OFF
  CM_SECURITY_USER_IS_WORKING
  CM_SECURITY_OLD_PSWORD_WRONG
  CM_SECURITY_NEW_PSWORD_WRONG
  CM_SECURITY_NO_CARD_READER
  CM_SECURITY_CARD_INFO_WRONG
  CM_SECURITY_NO_RESOURCES
  CM_SECURITY_NO_KERBEROS_SUPPORT
  CM_SECURITY_TAC_KEY_MISSING
  CM_SECURITY_PWD_EXPIRED_NO_RETRY
  CM_SECURITY_COMPLEXITY_ERROR
  CM_SECURITY_PASSWORD_TOO_SHORT
  CM_SECURITY_UPD_PSWORD_WRONG
  CM_SECURITY_TA_RECOVERY
  CM_SECURITY_PROTOCOL_CHANGED
  CM_SECURITY_SHUT_WARN
  CM_SECURITY_ENC_LEVEL_TOO_HIGH
  CM_SECURITY_PWD_EXPIRED_RETRY
  CM_SECURITY_USER_GLOBALLY_UNKNOWN
  CM_SECURITY_USER_SIGNED_ON_OTHER_NODE
  CM_SECURITY_TRANSIENT_ERROR

– Bei allen anderen Fehlersituationen ändert das Programm seinen Zustand nicht.

Hinweis

- Bei einem *Receive_Mapped_Data*-Aufruf kann ein Programm nur so viele Daten empfangen, wie im Parameter *requested_length* angegeben wurde. Es ist deshalb möglich, dass das Programm damit noch nicht die komplette Nachricht, die vom Partner gesendet wurde, gelesen hat. Dem Parameter *data_received* können Sie entnehmen, ob noch weitere Daten der Nachricht gelesen werden müssen.
  - Falls das Programm bereits die komplette Nachricht empfangen hat, hat der Parameter *data_received* den Wert CM_COMPLETE_DATA_RECEIVED.
  - Hat das Programm noch nicht alle Daten der Nachricht empfangen, hat der Parameter *data_received* den Wert CM_INCOMPLETE_DATA_RECEIVED. Um die restlichen Daten der Nachricht zu lesen, müssen solange *Receive_Mapped_Data*- bzw. *Receive*-Aufrufe abgesetzt werden, bis *data_received* den Wert CM_COMPLETE_DATA_RECEIVED hat.
- Wurde vor einem blockierenden *Receive_Mapped_Data*-Aufruf mit dem Aufruf *Set_Receive_Timer* eine maximale Wartezeit eingestellt, dann kehrt der Programmlauf spätestens nach Ablauf der Wartezeit aus dem *Receive_Mapped_Data*-Aufruf zurück und der *Receive_Mapped_Data*-Aufruf liefert dann das Ergebnis (*return_code)* CM_OPERATION_INCOMPLETE zurück.
- Mit einem einzigen Aufruf kann ein Programm sowohl Daten als auch das Senderecht empfangen. Die Parameter *return_code*, *data_received* und *status_received* geben Auskunft über die Art der Information, die ein Programm erhalten hat.
- Falls das Programm den *Receive_Mapped_Data*-Aufruf im Zustand "Send" absetzt, wird das Senderecht an den UTM-Vorgang abgegeben. Auf diese Weise wird die Senderichtung der Conversation geändert.
- Ein *Receive*-Aufruf mit *requested_length*=0 hat keine spezielle Bedeutung.
  Falls Daten vorliegen, werden diese in der Länge 0 empfangen mit *data_received*=CM_INCOMPLETE_DATA_RECEIVED.
  Falls keine Daten vorliegen, kann das Senderecht empfangen werden. D.h. in diesem Fall können entweder Daten oder das Senderecht empfangen werden, aber nicht beides.
- Falls eine Teilnachricht mit mehreren *Receive_Mapped_Data*-Aufrufen empfangen wird (*data_received* hat den Wert CM_INCOMPLETE_DATA_RECEIVED außer beim letzten *Receive_Mapped_Data*-Aufruf), so werden die Parameter *map_name* und *map_name_length* nur beim ersten Aufruf von *Receive_Mapped_Data* versorgt. Sie werden bei den folgenden *Receive_Mapped_Data*-Aufrufen aber nicht überschrieben.
- Übergibt die UTM-Partner-Anwendung ein leeres Formatkennzeichen (d.h. 8 Blanks), dann wird *map_name* mit 8 Leerzeichen belegt und *map_name_length*=-1 gesetzt.

Verhalten im Fehlerfall

CM_RESOURCE_FAILURE_RETRY
Conversation neu einrichten. Tritt der Fehler häufiger auf, ist eventuell der Pagepool der UTM-Anwendung zu klein dimensioniert und sollte vergrößert werden (MAX-Anweisung, PGPOOL=).

CM_RESOURCE_FAILURE_NO_RETRY
Systemdienst informieren und Diagnoseunterlagen erstellen. Es kann auch eine Störung im Transportsystem die Ursache für diesen Fehlercode sein.

CM_PROGRAM_STATE_CHECK
Programm ändern.

CM_PROGRAM_PARAMETER_CHECK
Programm ändern.

CM_PRODUCT_SPECIFIC_ERROR
Programm ändern.

CM_MAP_ROUTINE_ERROR
Programm ändern.

CM_OPERATION_INCOMPLETE
Conversation und Kommunikationsverbindung müssen explizit mit dem Aufruf *Disable_UTM_UPIC* abgebaut werden.
Jeder andere Aufruf kann zu unvorhersehbaren Ergebnissen führen.

CM_SECURITY_USER_UNKNOWN
Die UTM-Benutzerkennung ist nicht generiert. Benutzerkennung verwenden, die generiert ist oder gewünschte Benutzerkennung generieren oder dynamisch konfigurieren.

CM_SECURITY_STA_OFF
Benutzerkennung mit STATUS=ON generieren oder per Administration entsperren.

CM_SECURITY_USER_IS_WORKING
Andere UTM-Benutzerkennung benutzen oder den Vorgang des bereits angemeldeten Benutzers beenden.

CM_SECURITY_OLD_PSWORD_WRONG
Passwort korrekt angegeben.

CM_SECURITY_NEW_PSWORD_WRONG
Altes Passwort bis Ablauf der Gültigkeitsdauer weiterverwenden.

CM_SECURITY_NO_CARD_READER
Der Benutzer ist mit Magnetstreifenkarte generiert und kann sich nicht über UPIC anmelden.

CM_SECURITY_CARD_INFO_WRONG
Der Benutzer ist mit Chipkarte generiert.

CM_SECURITY_NO_RESOURCES
Später wieder probieren.

CM_SECURITY_NO_KERBEROS_SUPPORT
Der Benutzer ist mit einem Kerberos-Prinzipal generiert und kann sich nicht über UPIC anmelden.

CM_SECURITY_TAC_KEY_MISSING
Generierung oder Programm ändern.

CM_SECURITY_PWD_EXPIRED_NO_RETRY
Die Gültigkeitsdauer des Passworts ist abgelaufen. Das Passwort muss per Administration geändert werden, bevor der Benutzer sich wieder anmelden kann.

CM_SECURITY_COMPLEXITY_ERROR
Das neue Passwort entsprechend den Anforderungen der generierten Komplexitätsstufe wählen, siehe KDCDEF-Anweisung USER PROTECT-PW=.

CM_SECURITY_PASSWORD_TOO_SHORT
Neues längeres Passwort verwenden oder Generierung ändern, siehe KDCDEF-Anweisung USER PROTECT-PW= *length*, ... (Wert für die minimale Länge).

CM_SECURITY_UPD_PSWORD_WRONG
Das Passwort entspricht nicht der geforderten Komplexitätsstufe oder hat nicht die erforderliche Länge, siehe KDCDEF-Anweisung USER PROTECT-PW=. Das Passwort muss per Administration geändert werden, bevor sich der Benutzer wieder anmelden kann.

CM_SECURITY_TA_RECOVERY
Für die angegebene Benutzerkennung ist ein Transaktionswiederanlauf erforderlich.

CM_SECURITY_PROTOCOL_CHANGED
Der Benutzer hat einen offenen Vorgang, der nicht von einem UPIC-Client aus fortgesetzt werden kann.

CM_SECURITY_SHUT_WARN
Die UTM-Anwendung wird beendet; es dürfen sich nur noch Benutzer mit Administrationsberechtigung anmelden. Abwarten, bis die Anwendung neu gestartet wurde.

CM_SECURITY_ENC_LEVEL_TOO_HIGH
Auf der Verbindung ist der für die Fortsetzung des offenen Vorgangs nötige Verschlüsselungsmechanismus nicht verfügbar.

CM_SECURITY_PWD_EXPIRED_RETRY
Den Aufbau der Conversation mit Angabe des alten und eines neuen Passworts wiederholen.

Die folgenden sekundären Returncodes treten nur im Zusammenhang mit UTM-Cluster-Anwendungen auf:

CM_SECURITY_USER_GLOBALLY_UNKNOWN
Die angegebene Benutzerkennung ist in der Cluster-User-Datei nicht bekannt.

CM_SECURITY_USER_SIGNED_ON_OTHER_NODE
Mit dieser Benutzerkennung hat sich bereits ein Benutzer an einer anderen Knoten-Anwendung angemeldet.

CM_SECURITY_TRANSIENT_ERROR
Beim Anmelden trat ein temporärer Fehler auf. Auf die Cluster-User-Datei konnte innerhalb der in der Knoten-Anwendung konfigurierten Zeit nicht zugegriffen werden.

Anmeldung später noch einmal versuchen.

Funktionsdeklaration: Receive_Mapped_Data

```
CM_ENTRY Receive_Mapped_Data (unsigned char CM_PTR  conversation_ID,
                              unsigned char CM_PTR  map_name,
                                    CM_INT32 CM_PTR  map_name_length,
                              unsigned char CM_PTR  buffer,
                                    CM_INT32 CM_PTR  requested_length,
                      CM_DATA_RECEIVED_TYPE CM_PTR  data_received,
                                    CM_INT32 CM_PTR  received_length,
                         CM_STATUS_RECEIVED CM_PTR  status_received,
            CM_CONTROL_INFORMATION_RECEIVED CM_PTR  request_to_send_received,
                             CM_RETURN_CODE CM_PTR  return_code )
```

Send_Data - Daten an einen UTM-Service senden

Mit dem Aufruf *Send_Data* (CMSEND) sendet ein Programm Daten an einen UTM-Vorgang. Jedesmal nachdem ein Programm das Senderecht erhalten hat, muss es einen *Send_Data*- oder einen *Send_Mapped_Data*-Aufruf absetzen. Dies ist der Fall

- unmittelbar nach einem erfolgreichen *Allocate*-Aufruf
- wenn nach dem *Receive*- bzw. *Receive_Mapped_Data*-Aufruf die Characteristic *status_received* den Wert CM_SEND_RECEIVED hat (d.h. wenn das Programm das Senderecht empfangen hat).

Syntax

CMSEND (conversation_ID, buffer, send_length, control_information_received,
return_code)

Parameter

→ conversation_ID Identifikation der Conversation

→ buffer Puffer mit den zu sendenden Daten. Die Länge der Daten wird im Parameter *send_length* angegeben.

→ send_length Länge der zu sendenden Daten in Bytes.

Minimum: 0, Maximum: 32767

Ein *Send_Data*-Aufruf mit der Länge 0 bewirkt, dass eine Nachricht der Länge 0 gesendet wird.

← control_information_received
Wird nur syntaktisch unterstützt und kann nur den Wert CM_REQ_TO_SEND_NOT_RECEIVED annehmen.

Der Wert in *control_information_received* ist undefiniert, wenn das Ergebnis in *return_code* ungleich CM_OK ist.

← return_code Ergebnis des Funktionsaufrufs

Ergebnis (*return_code*)

CM_OK
: Aufruf ok

CM_TPN_NOT_RECOGNIZED
: Dieser Returncode kann nur beim ersten *Send_Data*-Aufruf nach einem *Allocate*-Aufruf auftreten. Nach dem Einrichten der Conversation ist ein Fehler aufgetreten, der zur Beendigung der Conversation führte.

CM_DEALLOCATED_ABEND
: mögliche Ursachen:
  - UTM-Anwendungsende
  - Verbindungsabbau durch UTM-Administration
  - Verbindungsabbau durch das Transportsystem

CM_RESOURCE_FAILURE_RETRY
: Ein vorübergehender Betriebsmittelengpass führte zur Beendigung der Conversation. Möglicherweise können im UTM-Pagepool keine Daten mehr zwischengespeichert werden.

  Maßnahme: UTM-Pagepool vergrößern (MAX-Anweisung, PGPOOL=).

CM_PROGRAM_STATE_CHECK
: Der Aufruf ist im aktuellen Zustand nicht erlaubt.

CM_PROGRAM_PARAMETER_CHECK
: Der Wert in *conversation_ID* ist ungültig oder der Wert in *send_length* ist größer als 32767 oder kleiner als 0.

Zustandsänderung

Falls das Ergebnis CM_OK ist, bleibt das Programm im Zustand "Send".

Beim Ergebnis CM_TPN_NOT_RECOGNIZED, CM_DEALLOCATED_ABEND oder CM_RESOURCE_FAILURE_RETRY/NO_RETRY geht das Programm in den Zustand "Reset" über.

Bei allen anderen Fehlersituationen ändert das Programm seinen Zustand nicht.

Hinweis

UPIC puffert die zu sendenden Daten und schickt sie erst zu einem späteren Zeitpunkt an den UTM-Server. Aus diesem Grund kann es passieren, dass eine Beendigung der UTM-Anwendung nicht unmittelbar, sondern erst bei einem Folgeaufruf als Ergebnis geliefert wird.

Verhalten im Fehlerfall

CM_RESOURCE_FAILURE_RETRY
Conversation neu einrichten.

CM_PROGRAM_STATE_CHECK
Programm ändern.

CM_PROGRAM_PARAMETER_CHECK
Programm ändern.

Funktionsdeklaration: Send_Data

```
CM_ENTRY Send_Data ( unsigned char CM_PTR  conversation_ID,
                     unsigned char CM_PTR  buffer,
                          CM_INT32 CM_PTR  send_length,
    CM_CONTROLINFORMATION_RECEIVED CM_PTR  control_information_received,
                    CM_RETURN_CODE CM_PTR  return_code )
```

Send_Mapped_Data - Daten und Formatkennzeichen senden

Mit dem Aufruf *Send_Mapped_Data* (CMSNDM) sendet ein Programm Daten und ein Formatkennzeichen an einen UTM-Vorgang. Jedesmal nachdem ein Programm das Senderecht erhalten hat, muss es einen *Send_Data*- oder *Send_Mapped_Data*-Aufruf absetzen. Dies ist der Fall

- unmittelbar nach einem erfolgreichen *Allocate*-Aufruf oder
- wenn nach dem *Receive*- bzw. *Receive_Mapped_Data*-Aufruf die Characteristic *status_received* den Wert CM_SEND_RECEIVED hat (d.h. wenn das Programm das Senderecht empfangen hat).

Syntax

CMSNDM (conversation_ID, map_name, map_name_length, buffer, send_length,
control_information_received, return_code)

Parameter

→ conversation_ID Identifikation der Conversation

→ map_name Formatkennzeichen, das an die UTM-Anwendung gesendet wird. Das Formatkennzeichen spezifiziert die Strukturierungsmerkmale für den Empfänger der Daten.

→ map_name_length Länge des Formatkennzeichens in Byte.

→ buffer Adresse des Puffers mit den zu sendenden Daten. Die Länge der Daten wird im Parameter *send_length* angegeben.

→ send_length Länge der zu sendenden Daten in Byte.

Minimum: 0, Maximum: 32767

Ein *Send_Mapped_Data*-Aufruf mit der Länge 0 bewirkt, dass eine Nachricht der Länge Null gesendet wird.

← control_information_received
Wird nur syntaktisch unterstützt und kann nur den Wert CM_REQ_TO_SEND_NOT_RECEIVED annehmen.

Der Wert in *control_information_received* ist undefiniert, wenn das Ergebnis in *return_code* ungleich CM_OK ist.

← return_code Ergebnis des Funktionsaufrufs

Ergebnis (*return_code*)

CM_OK
: Aufruf ok

CM_TPN_NOT_RECOGNIZED
: Dieser Returncode kann nur beim ersten *Send_Mapped_Data*-Aufruf nach einem *Allocate*-Aufruf auftreten. Nach dem Einrichten der Conversation ist ein Fehler aufgetreten, der zur Beendigung der Conversation führte.

CM_DEALLOCATED_ABEND
: mögliche Ursachen:
  - UTM-Anwendungsende
  - Verbindungsabbau durch UTM-Administration
  - Verbindungsabbau durch das Transportsystem

CM_RESOURCE_FAILURE_RETRY
: Ein vorübergehender Betriebsmittelengpass führte zur Beendigung der Conversation. Möglicherweise können im UTM-Pagepool keine Daten mehr zwischengespeichert werden.

CM_PROGRAM_STATE_CHECK
: Der Aufruf ist im aktuellen Zustand nicht erlaubt.

CM_PROGRAM_PARAMETER_CHECK
: Der Wert in *conversation_ID* ist ungültig oder der Wert in *send_length* ist größer als 32767 oder kleiner als Null.

CM_MAP_ROUTINE_ERROR
: mögliche Ursachen:
  - In der UTM-Partner-Anwendung werden keine Formatkennzeichen im UPIC-Protokoll unterstützt.
  - Die Länge des Formatkennzeichens ist kleiner 0 oder größer 8.

Zustandsänderung

- Falls das Ergebnis CM_OK ist, bleibt das Programm im Zustand "Send".
- Bei folgenden Ergebnissen geht das Programm in den Zustand "Reset" über:
  CM_TPN_NOT_RECOGNIZED
  CM_DEALLOCATED_ABEND
  CM_RESOURCE_FAILURE_RETRY/NO_RETRY
- Bei allen anderen Ergebnissen ändert das Programm seinen Zustand nicht.

Hinweis

- Die Daten werden immer transparent übertragen. Die gesendeten Daten werden dem Partner-UTM-Vorgang beim MGET-Aufruf angezeigt.
  Das Formatkennzeichen in *map_name* wird dem UTM-Vorgang im Feld KCMF/*kcfn* beim MGET-Aufruf übergeben.
- Aus Performancegründen puffert UPIC die zu sendenden Daten und schickt sie erst zu einem späteren Zeitpunkt (mit einem Folgeaufruf) an den UTM-Server. Aus diesem Grund kann es passieren, dass eine Beendigung der UTM-Anwendung nicht unmittelbar, sondern erst bei einem Folgeaufruf als Ergebnis geliefert wird.
- Sobald der Wert von *map_name* an openUTM gesendet wird, wird *map_name* zurückgesetzt.

Verhalten im Fehlerfall

CM_RESOURCE_FAILURE_RETRY
: Conversation neu einrichten. Tritt der Fehler häufiger auf, ist eventuell der Pagepool der UTM-Anwendung zu klein dimensioniert und sollte vergrößert werden (MAX-Anweisung, PGPOOL=).

CM_PROGRAM_STATE_CHECK
: Programm ändern.

CM_PROGRAM_PARAMETER_CHECK
: Programm ändern.

Funktionsdeklaration: Send_Mapped_Data

```
CM_ENTRY Send_Mapped_Data(unsigned char CM_PTR  conversation_ID,
                          unsigned char CM_PTR  map_name,
                               CM_INT32 CM_PTR  map_name_length,
                          unsigned_char CM_PTR  buffer,
                               CM_INT32 CM_PTR send_length,
        CM_CONTROL_INFORMATION_RECEIVED CM_PTR  control_information_received,
                         CM_RETURN_CODE CM_PTR  return_code )
```

Set_Allocate_Timer - Timer für den Allocate setzen

Der Aufruf *Set_Allocate_Timer*(CMSAT) setzt den Timeout für einen Allocate-Aufruf.

Wenn dieser Timer gesetzt ist, wird der Aufruf Allocate nach der im Feld *allocate_timer* festgelegten Zeit abgebrochen.

Der Aufruf *Set_Allocate_Timer* ist nur im Zustand „Init“ erlaubt.

Diese Funktion ist nicht Bestandteil der CPI-C-Spezifikation, sondern eine zusätzliche Funktion des UPIC-Trägersystems.



UPIC-Local: Der Aufruf *Set_Allocate_Timer* wird bei der Anbindung über UPIC-Local nicht unterstützt.

Syntax

CMSAT (conversation_ID, allocate_timer, return_code)

Parameter

→ conversation_ID Identifikation der Conversation.

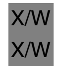
→ allocate_timer Zeit in Millisekunden, nach der ein Allocate-Aufruf unterbrochen wird. Der Allocate-Timer wird zurückgesetzt, wenn Sie *allocate_timer* auf 0 setzen. Die Wartezeit des Allocate-Aufrufs wird dann nicht mehr überwacht.

Der für *allocate_timer* angegebene Wert wird auf die nächste volle Sekunde aufgerundet.

← return_code Ergebnis des Funktionsaufrufs.

Ergebnis (*return_code*)

CM_OK
Aufruf ok



CM_CALL_NOT_SUPPORTED
Die Funktion wird nicht unterstützt. Dieser Returncode tritt nur bei UPIC-L auf.

CM_PROGRAM_STATE_CHECK
Die Conversation ist nicht im Zustand "Init"

CM_PROGRAM_PARAMETER_CHECK
Der Wert der *conversation_ID* ist ungültig oder in *allocate_timer* wurde ein Wert < 0 angegeben.

CM_PRODUCT_SPECIFIC_ERROR
Die UPIC-Instanz konnte nicht gefunden werden.

Zustandsänderung

Im fehlerfreien Fall liefert die Funktion das Ergebnis CM_OK zurück. Der Aufruf ändert den Zustand der Conversation nicht.

Hinweis

- Der *Set_Allocate_Timer* ist nur sinnvoll im Zusammenhang mit dem Allocate-Aufruf. *Set_Allocate_Timer* kann zwischen einem *Initialize_Conversation*- und einem Allocate-Aufruf beliebig oft aufgerufen werden. Es gilt immer der Wert, der beim letzten Aufruf von *Set_Allocate_Timer* vor einem Allocate-Aufruf gesetzt wurde.

Verhalten im Fehlerfall



CM_CALL_NOT_SUPPORTED
Muss nicht unbedingt ein Fehler sein: Falls eine Anwendung sowohl für UPIC-L als auch für UPIC-R vorgesehen ist, bedeutet dieser Returncode lediglich, dass die Anwendung mit einer UPIC-L-Bibliothek gebunden ist. In diesem Fall sind Timer-Funktionen nicht möglich. Das Programm kann sich diesen Returncode merken und auf weitere Aufrufe bzgl. des Timers verzichten.

CM_PROGRAM_STATE_CHECK
Programm ändern.

CM_PROGRAM_PARAMETER_CHECK
Programm ändern.

CM_PRODUCT_SPECIFIC_ERROR
Das Betriebssystem kann nicht genügend Speicherplatz für interne Puffer bereitstellen. Überprüfen Sie Ihr Programm auf zu hohe Speicherplatzanforderung und starten Sie ggf. Ihr System neu.

Funktionsdeklaration: Set_Allocate_Timer

```
CM_ENTRY Set_Allocate_Timer ( unsigned char CM_PTR  conversation_ID,
                                     CM_TIMEOUT CM_PTR  allocate_timer,
                                 CM_RETURN_CODE CM_PTR  return_code )
```

Set_Client_Context - Client-Kontext setzen

Der Aufruf *Set_Client_Context* (CMSCC) setzt den Wert für den Client-Kontext. Um den Wiederanlauf auf Client-Seite zu erleichtern, kann der Client einen von ihm selbst spezifizierten, sogenannten Client-Kontext bei openUTM hinterlegen. Immer wenn der Client Benutzerdaten an die UTM-Partner-Anwendung sendet, wird auch der letzte mit der Funktion *Set_Client_Context* gesetzte Client-Kontext an die UTM-Anwendung gesendet. Der Kontext wird von openUTM bis zum Ende der Conversation gesichert, falls er nicht durch einen neuen Kontext überschrieben wird.

Wird vom Client ein Wiederanlauf gefordert, so wird der zuletzt gesicherte Kontext zusammen mit der letzten Dialog-Nachricht an den Client zurück übertragen.

Der Client-Kontext wird von openUTM nur gesichert, wenn der Client über eine UTM-Benutzerkennung mit Restartfunktionalität angemeldet ist, da nur in diesem Fall ein Vorgangswiederanlauf möglich ist. In allen anderen Fällen wird der Kontext ignoriert.

Der Aufruf *Set_Client_Context* ist nur im Zustand "Send" erlaubt.

Diese Funktion ist nicht Bestandteil der CPI-C Spezifikation, sondern eine zusätzliche Funktion des UPIC-Trägersystems.

Syntax

CMSCC (conversation_ID, client_context, client_context_length, return_code)

Parameter

→ conversation_ID   Identifikation der Conversation.

→ client_context   gibt den Kontext an, den der Client an openUTM senden will.

→ client_context_length
    Länge des Kontexts.
    Minimum 0, Maximum: 8

← return_code   Ergebnis des Funktionsaufrufs.

Ergebnis (*return_code*)

CM_OK
: Aufruf ok

CM_CALL_NOT_SUPPORTED
: Die Funktion wird nicht unterstützt. Dieser Returncode tritt auf, wenn kein Client-Kontext eingesetzt werden kann.

CM_PROGRAM_STATE_CHECK
: Die Conversation ist nicht im Zustand "Send".

CM_PROGRAM_PARAMETER_CHECK
: Der Wert in *conversation_ID* ist ungültig oder der Wert von *client_context_length* ist kleiner als 0 oder größer als 8.

CM_PRODUCT_SPECIFIC_ERROR
: Die UPIC-Instanz konnte nicht gefunden werden.

Zustandsänderung

Im fehlerfreien Fall liefert die Funktion das Ergebnis CM_OK zurück. Der Aufruf ändert den Zustand der Conversation nicht.

Hinweis

- Falls der Returncode von CM_OK verschieden ist, bleibt *client_context* unverändert.
- Der interne Puffer für den Client-Kontext ist derzeit auf 8 Bytes beschränkt.

Verhalten im Fehlerfall

CM_CALL_NOT_SUPPORTED
: Ist nicht unbedingt ein Fehler. Falls eine UPIC-R Anwendung mit verschiedenen UTM-Partnern kommuniziert, bedeutet dieser Returncode lediglich, dass die Anwendung mit einer UTM-Anwendung kommuniziert, die keinen Client-Kontext empfangen kann. Das Programm kann sich diesen Returncode merken und auf weitere Aufrufe bzgl. Client-Kontext verzichten.

CM_PROGRAM_STATE_CHECK
: Programm ändern.

CM_PROGRAM_PARAMETER_CHECK
: Programm ändern.

CM_PRODUCT_SPECIFIC_ERROR
Das Betriebssystem kann nicht genügend Speicherplatz für interne Puffer bereitstellen. Überprüfen Sie Ihr Programm auf zu hohe Speicherplatzanforderung und starten Sie ggf. Ihr System neu.

Funktionsdeklaration: Set_Client_Context

```
CM_ENTRY Set_Client_Context (
                        unsigned char    CM_PTR  conversation_ID,
                        unsigned char    CM_PTR  client_context,
                        CM_INT32         CM_PTR  client_context_length,
                        CM_RETURN_CODE   CM_PTR  return_code )
```

Set_Communication_Protocol – Setzen der Conversation Characteristic COMMUNICATION_PROTOCOL

Der Aufruf *Set_Communication_Protocol* (CMSCP) setzt für die Conversation die Characteristic *COMMUNICATION_PROTOCOL.*

Set_Communication_Protocol ändert die Werte, die beim *Initialize_Conversation*-Aufruf aus der Side Information entnommen wurden. Die geänderten Werte gelten nur für die Dauer einer Conversation; die Werte in der Side Information selbst werden nicht verändert.

Der *Set_Communication_Protocol*-Aufruf darf nach *Allocate* nicht mehr ausgeführt werden.

Diese Funktion gehört zu den zusätzlichen Funktionen des Trägersystems UPIC; sie ist nicht Bestandteil der CPI-C-Schnittstelle.

Syntax

CMSCP (conversation_ID, protocol, return_code)

Parameter

→ conversation_ID Identifikation der Conversation

→ protocol legt fest, welches Kommunikationsprotokoll zwischen UPIC-Client und UTM-Server benutzt werden soll.

Für *protocol* können Sie folgende Werte angeben:

CM_COMMUNICATION_PROTOCOL_34
Der UPIC-Client versucht, die Verbindung zum UTM-Server auf Basis des Protokolls 34 aufzubauen, z.B. bei UTM-Anwendungen mit Version < 4.0.

CM_COMMUNICATION_PROTOCOL_40
Der UPIC-Client versucht, die Verbindung zum UTM-Server auf Basis des Protokolls 40 aufzubauen.

← return_code Ergebnis des Funktionsaufrufes

Ergebnis (*return_code*)

CM_OK
: Aufruf OK

CM_PROGRAM_PARAMETER_CHECK
: Der Wert in *conversation_ID* oder die Protokollversion in *protocol* ist ungültig.

CM_PRODUCT_SPECIFIC_ERROR
: Die UPIC-Instanz konnte nicht gefunden werden.

CM_PROGRAM_STATE_CHECK
: Die Conversation ist nicht im Zustand “Initialize”

Zustandsänderung

Der Aufruf ändert den Zustand der Conversation nicht.

Hinweis

Falls der Returncode von CM_OK verschieden ist, bleibt die Characteristic unverändert.

Verhalten im Fehlerfall

CM_PROGRAM_STATE_CHECK
: Programm ändern

CM_PROGRAM_PARAMETER_CHECK
: Programm ändern

CM_PRODUCT_SPECIFIC_ERROR
: Das Betriebssystem kann nicht genügend Speicherplatz für interne Puffer bereitstellen. Überprüfen Sie Ihr Programm auf zu hohe Speicherplatzanforderung und starten Sie ggf. Ihr System neu.

Funktionsdeklaration: Set_Communication_Protocol

```
CM_ENTRY Set_Communication_Protocol(
                unsigned char                 CM_PTR  conversation_ID,
                CM_COMMUNICATION_PROTOCOL     CM_PTR  protocol,
                CM_RETURN_CODE                CM_PTR  return_code )
```

Set_Conversation_Encryption_Level - Verschlüsselungsebene setzen

Der Aufruf *Set_Conversation_Encryption_Level* (CMSCEL) beeinflusst den Wert für die Conversation Characteristic *ENCRYPTION-LEVEL*. Mit der Verschlüsselungsebene wird festgelegt, ob während der Conversation die Benutzerdaten verschlüsselt oder unverschlüsselt übertragen werden sollen. Der Aufruf überschreibt den Wert von *encryption_level*, der beim *Initialize_Conversation*-Aufruf zugewiesen wurde.

Der Aufruf *Set_Conversation_Encryption_Level* ist nur im Zustand "Initialize" erlaubt.

UPIC-Local: Die Datenübertragung ist durch die Art der Übertragung selbst geschützt. Der Aufruf *Set_Conversation_Encryption_Level* wird nicht unterstützt.

Diese Funktion gehört zu den zusätzlichen Funktionen des Trägersystems UPIC; sie ist nicht Bestandteil der CPI-C-Schnittstelle.

Syntax

CMSCEL (conversation_ID, encryption_level, return_code)

Parameter

→ conversation_ID Identifikation der Conversation

→ encryption_level legt fest, ob in der Conversation die Benutzerdaten verschlüsselt oder nicht verschlüsselt werden sollen. Folgende Werte können Sie angeben:

CM_ENC_LEVEL_NONE
Die Benutzerdaten der Conversation sollen unverschlüsselt übertragen werden.

CM_ENC_LEVEL_1
Die Benutzerdaten sollen verschlüsselt übertragen werden, zum Verschlüsseln wird der DES-Algorithmus benutzt. Für den Austausch des DES-Schlüssels wird ein RSA-Schlüssel mit einer Schlüssellänge von 200 bit verwendet.

CM_ENC_LEVEL_2
Die Benutzerdaten sollen verschlüsselt übertragen werden, zum Verschlüsseln wird der AES-Algorithmus benutzt. Für den Austausch des AES-Schlüssels wird ein RSA-Schlüssel mit einer Schlüssellänge von 512 bit verwendet.

CM_ENC_LEVEL_3
Die Benutzerdaten sollen verschlüsselt übertragen werden, zum Verschlüsseln wird der AES-Algorithmus benutzt. Für den Austausch des AES-Schlüssels wird ein RSA-Schlüssel mit einer Schlüssellänge von 1024 bit verwendet.

CM_ENC_LEVEL_4
Die Benutzerdaten sollen verschlüsselt übertragen werden, zum Verschlüsseln wird der AES-Algorithmus benutzt. Für den Austausch des AES-Schlüssels wird ein RSA-Schlüssel mit einer Schlüssellänge von 2048 bit verwendet.

← return_code Ergebnis des Funktionsaufrufs.

Ergebnis (***return_code***)

CM_OK
Aufruf ok



CM_CALL_NOT_SUPPORTED
Die Funktion wird nicht unterstützt. Dieser Returncode tritt nur bei UPIC-L auf. Er zeigt dem Programm an, dass keine Verschlüsselung notwendig ist.

CM_PROGRAM_STATE_CHECK
Die Conversation ist nicht im Zustand "Init".

CM_PROGRAM_PARAMETER_CHECK
Der Wert der *conversation_ID* ist ungültig oder der Wert von *encryption_level* ist undefiniert.

CM_PRODUCT_SPECIFIC_ERROR
Die UPIC-Instanz konnte nicht gefunden werden.

CM_ENCRYPTION_NOT_SUPPORTED
Für diese Conversation ist keine Verschlüsselung möglich, weil entweder
- das Zusatzprodukt openUTM-Crypt nicht installiert ist.
- der UPIC-Client nicht verschlüsseln kann, weil das Produkt openUTM-Client ohne die Lizenz zum Verschlüsseln installiert wurde.
- die UTM-Partner-Anwendung keine Verschlüsselung will, da der UPIC-L-Client vertrauenswürdig (trusted) ist.



CM_ENCRYPTION_LEVEL_NOT_SUPPORTED
die Verschlüsselung mit der angegebenen Verschlüsselungsebene (*encryption_level*) wird von UPIC nicht unterstützt.

Zustandsänderung

Im fehlerfreien Fall liefert die Funktion das Ergebnis CM_OK zurück. Der Aufruf ändert den Zustand der Conversation nicht.

Hinweis

- Falls der Returncode von CM_OK verschieden ist, bleibt die Characteristic *ENCRYPTION_LEVEL* unverändert.
- Ist die Verschlüsselungsebene, die von der UTM-Anwendung gefordert wird, höher als die auf der UPIC-Client Seite, wird die höhere Verschlüsselungsebene wirksam. D.h. wenn die UTM-Anwendung eine bestimmte Verschlüsselungsebene fordert, so verschlüsselt der UPIC-Client die Daten mit dieser Stufe, ungeachtet der von der UPIC-Anwendung eingestellten Verschlüsselungsebene.
- Wenn zum Zeitpunkt des Aufrufs keine Kommunikationsverbindung zu einer UTM-Partner-Anwendung besteht, beendet sich die Funktion immer mit dem Returncode CM_OK. Erst beim folgenden *Allocate*-Aufruf wird entschieden, ob die gewünschte Verschlüsselungsebene wirksam wird.

Verhalten im Fehlerfall



CM_CALL_NOT_SUPPORTED
: Muss kein Fehler sein: Falls eine Anwendung sowohl für UPIC-L als auch für UPIC-R vorgesehen ist, bedeutet dieser Returncode lediglich, dass die Anwendung mit einer UPIC-L-Bibliothek gebunden ist. In diesem Fall ist Verschlüsselung nicht nötig. Das Programm kann sich diesen Returncode merken und auf weitere Aufrufe zur Verschlüsselung verzichten.

CM_PROGRAM_STATE_CHECK
: Programm ändern.

CM_PROGRAM_PARAMETER_CHECK
: Programm ändern.

CM_PRODUCT_SPECIFIC_ERROR
: Das Betriebssystem kann nicht genügend Speicherplatz für interne Puffer bereitstellen. Überprüfen Sie Ihr Programm auf zu hohe Speicherplatzanforderung und starten Sie ggf. Ihr System neu.

CM_ENCRYPTION_NOT_SUPPORTED
Muss kein Fehler sein: Falls eine UPIC-R Anwendung mit verschiedenen UTM-Partnern kommuniziert, von denen einige verschlüsseln können und andere nicht, bedeutet dieser Returncode lediglich, dass die Anwendung mit einer UTM-Anwendung kommuniziert, die nicht verschlüsseln kann oder will. In diesem Fall ist Verschlüsselung nicht möglich. Das Programm kann sich diesen Returncode merken und auf weitere Aufrufe zur Verschlüsselung verzichten.

CM_ENCRYPTION_LEVEL_NOT_SUPPORTED
Die UPIC-Bibliothek hat eventuell eine alte Encryption-Bibliothek geladen. Stellen Sie sicher, dass die Encryption-Bibliothek der neuesten openUTM-Client Version installiert ist und auch geladen wird. Beachten Sie bitte die Suchreihenfolge für Bibliotheken in den verschiedenen Betriebssystemen.

Funktionsdeklaration: Set_Conversation_Encryption_Level

```
CM_ENTRY Set_Conversation_Encryption_Level
                           unsigned char CM_PTR  conversation_ID,
                     CM_ENCRYPTION_LEVEL CM_PTR  encryption_level,
                          CM_RETURN_CODE CM_PTR  return_code )
```

Set_Conversation_Security_New_Password - neues Passwort setzen

Der Aufruf *Set_Conversation_Security_New_Password* (CMSCSN) setzt den Wert für die Characteristics *security_new_password* und *security_new_password_length* der Conversation. Unter dem *security_new_password* versteht man das neue Passwort einer UTM-Benutzerkennung.

Ein Programm kann ein neues Passwort nur dann angeben, wenn die Characteristics *security_type* auf CM_SECURITY_PROGRAM gesetzt ist.

Der Aufruf darf nach *Allocate* nicht mehr ausgeführt werden.

Diese Funktion gehört zu den zusätzlichen Funktionen des Trägersystems UPIC; sie ist nicht Bestandteil der CPI-C-Schnittstelle.

Syntax

CMSCSN (conversation_ID, security_new_password, security_new_password_length, return_code)

Parameter

→ conversation_ID Identifikation der Conversation.

→ security_new_password
Passwort, das das alte Passwort ersetzen soll. Die UTM-Partner-Anwendung verwendet dieses Passwort, um nach gültiger Zugangsberechtigung mit dem alten Passwort das alte Passwort durch dieses neue Passwort zu ersetzen.

→ security_new_password_length
Länge des in *security_new_password* angegebenen Passwort in Byte. Minimum: 0, Maximum: 8.

Wird hier 0 angegeben, dann wird *security_new_password* mit 8 Leerzeichen belegt, d.h. openUTM ändert das bestehende Passwort nicht.

← return_code Ergebnis des Funktionsaufrufs.

Ergebnis (*return_code*)

CM_OK
: Aufruf ok

CM_CALL_NOT_SUPPORTED
: Die Funktion wird nicht unterstützt.

CM_PROGRAM_STATE_CHECK
: Die Conversation ist nicht im Zustand "Initialize" oder *security_type* ist nicht auf CM_SECURITY_PROGRAM gesetzt.

CM_PROGRAM_PARAMETER_CHECK
: Der Wert der *conversation_ID* ist ungültig, der Wert in *security_new_password_length* ist kleiner als 0 oder größer als 8, oder das neue Passwort besteht nur aus Leerzeichen.

CM_PRODUCT_SPECIFIC_ERROR
: Die UPIC-Instanz konnte nicht gefunden werden.

Falls das Ergebnis nicht CM_OK ist, bleiben die Characteristics *security_new_password* und *security_new_password_length* unverändert.

Zustandsänderung

Der Aufruf ändert den Zustand der Conversation nicht.

Hinweis

- Wenn ein Programm *Set_Conversation_Security_New_Password* aufruft, muss auch eine Benutzerkennung angegeben werden. Die Benutzerkennung wird im Programm mit dem Aufruf *Set_Conversation_Security_User_ID* gesetzt.
- Ein ungültiges Passwort wird bei diesem Aufruf nicht entdeckt. Die Partner-Anwendung überprüft das Passwort nach dem Einrichten der Conversation auf Gültigkeit. Bei ungültigem Passwort schickt die Partner-Anwendung eine Fehlermeldung, die in der UPIC-Logging-Datei abgespeichert wird.
- Das Programm erkennt das fehlerhafte Passwort durch den Returncode CM_SECURITY_NOT_VALID. Dieser wird nach einem *Allocate* folgenden CPI-C-Aufruf zurückgegeben.

- Werden nacheinander mehrere Conversations zur gleichen Partner-Anwendung aufgebaut (d.h. die Kommunikationsverbindung wird nicht jedesmal auf- und abgebaut), so kann das Ergebnis von CMSCSN nach dem ersten CMINIT CM_OK, nach allen folgenden CMINIT-Aufrufen aber CM_CALL_NOT_SUPPORTED sein. Die UPIC-Bibliothek baut erst nach dem ersten CMALLC-Aufruf eine Verbindung zur Partner-Anwendung auf und kann erst dann feststellen, ob die Version der Partner-Anwendung Passwortänderungen unterstützt.
  Das Programm erkennt nach dem ersten CMSCSN-Aufruf mit dem Ergebnis CM_OK die fehlende Unterstützung für Passwortänderungen erst durch den Returncode CM_SECURITY_NOT_SUPPORTED.
  Dieser wird nach einem Allocate-Aufruf zurückgegeben.
- Wenn für das neue Passwort nur Leerzeichen angegeben werden, so bedeutet dies, dass die UTM-Anwendung das Passwort zurücksetzen sollte, d.h. der Benutzer benötigt kein Passwort mehr. Vom Client aus ist das aber nicht erlaubt, daher wird der Fehler CM_PROGRAM_PARAMETER_CHECK zurückgegeben.

Verhalten im Fehlerfall

CM_PROGRAM_PARAMETER_CHECK
: Programm ändern.

CM_PRODUCT_SPECIFIC_ERROR
: Das Betriebssystem kann nicht genügend Speicherplatz für interne Puffer bereitstellen. Überprüfen Sie Ihr Programm auf zu hohe Speicherplatzanforderung und starten Sie ggf. Ihr System neu.

Funktionsdeklaration: Set_Conversation_Security_New_Password

```
CM_ENTRY Set_Conversation_Security_New_Password (
                          unsigned char CM_PTR  conversation_ID,
                          unsigned char CM_PTR  security_new_password,
                               CM_INT32 CM_PTR  security_new_password_length,
                         CM_RETURN_CODE CM_PTR  return_code )
```

Set_Conversation_Security_Password - Passwort setzen

Die Funktion *Set_Conversation_Security_Password* (CMSCSP) setzt die Werte für die Characteristics *security_password* und *security_password_length* der Conversation. Unter dem *security_password* versteht man das Passwort einer UTM-Benutzerkennung.

Ein Programm kann ein Passwort nur dann angeben, wenn die Characteristic *security_type* auf CM_SECURITY_PROGRAM gesetzt ist.

Der Aufruf darf nach *Allocate* nicht mehr ausgeführt werden.

Diese Funktion gehört zu den Advanced Functions.

Syntax

CMSCSP (conversation_ID, security_password, security_password_length, return_code)

Parameter

→ conversation_ID Identifikation der Conversation

→ security_password Passwort, das zum Einrichten der Conversation benutzt wird. Die UTM-Partner-Anwendung verwendet dieses Passwort samt der Benutzerkennung, um die Zugangsberechtigung zu überprüfen.

Das Passwort wird im lokal auf der Maschine verwendeten Code angegeben. Falls erforderlich wird es nach EBCDIC konvertiert, siehe Abschnitt „Code-Konvertierung“ auf Seite 68f.

→ security_password_length
Länge des in *security_password* angegebenen Passworts in Byte.

Minimum: 0, Maximum: 8

Wird hier 0 angegeben, dann wird *security_password* mit 8 Leerzeichen belegt, d.h. für die Zugangsprüfung wird kein Passwort an openUTM übergeben.

← return_code Ergebnis des Funktionsaufrufs

Ergebnis (*return_code*)

CM_OK
: Aufruf ok

CM_PROGRAM_STATE_CHECK
: Die Conversation ist nicht im Zustand "Initialize" oder *security_type* ist nicht auf CM_SECURITY_PROGRAM gesetzt.

CM_PROGRAM_PARAMETER_CHECK
: Der Wert in *conversation_ID* ist ungültig oder der Wert in *security_password_length* ist kleiner als 0 oder größer als 8.

CM_PRODUCT_SPECIFIC_ERROR
: Die UPIC-Instanz konnte nicht gefunden werden.

Falls das Ergebnis nicht CM_OK ist, bleiben die Characteristics *security_password* und *security_password_length* unverändert.

Zustandsänderung

Keine Zustandsänderung.

Hinweis

- Wenn ein Programm *Set_Conversation_Security_Password* aufruft, muss auch eine Benutzerkennung angegeben werden. Die Benutzerkennung wird im Programm mit dem Aufruf *Set_Conversation_Security_User_ID* gesetzt.
- Ein ungültiges Passwort wird bei diesem Aufruf nicht entdeckt. Die Partner-Anwendung überprüft das Passwort nach dem Einrichten der Conversation auf Gültigkeit. Bei ungültigem Passwort schickt die Partner-Anwendung eine Fehlermeldung, die in der UPIC-Logging-Datei (siehe Abschnitt „UPIC-Logging-Datei“ auf Seite 337) abgespeichert wird.
- Das Programm erkennt das fehlerhafte Passwort durch den Returncode CM_SECURITY_NOT_VALID. Dieser wird nach einem dem *Allocate* folgenden CPI-C-Aufruf zurückgegeben.

Verhalten im Fehlerfall

CM_PROGRAM_PARAMETER_CHECK
: Programm ändern.

CM_PRODUCT_SPECIFIC_ERROR
: Das Betriebssystem kann nicht genügend Speicherplatz für interne Puffer bereitstellen. Überprüfen Sie Ihr Programm auf zu hohe Speicherplatzanforderung und starten Sie ggf. Ihr System neu.

Funktionsdeklaration: Set_ Conversation_Security_Password

```
CM_ENTRY Set_Conversation_Security_Password (
                              unsigned char CM_PTR conversation_ID,
                              unsigned char CM_PTR security_password,
                                   CM_INT32 CM_PTR  security_password_length,
                             CM_RETURN_CODE CM_PTR  return_code )
```

Set_Conversation_Security_Type - Security-Typ setzen

Die Funktion *Set_Conversation_Security_Type* (CMSCST) setzt den Wert für die Characteristic *security_type* der Conversation.

Der Aufruf überschreibt den Wert, der beim *Initialize_Conversation*-Aufruf zugewiesen wurde und darf nach *Allocate* nicht mehr ausgeführt werden.

Diese Funktion gehört zu den Advanced Functions.

Syntax

CMSCST (conversation_ID, security_type, return_code)

Parameter

→ conversation_ID Identifikation der Conversation

→ security_type gibt den Typ von Zugangsinformationen an, die beim Einrichten der Conversation an die Partner-Anwendung gesendet werden. Mit Hilfe dieser Informationen überprüft die Partner-Anwendung die Zugangsberechtigung.

Für *security_type* können folgende Werte gesetzt werden:

CM_SECURITY_NONE
Es werden keine Zugangsinformationen an die Partner-Anwendung übertragen.

CM_SECURITY_PROGRAM
Als Zugangsinformationen werden die Werte der Characteristics *security_user_ID* und *security_password* verwendet. D.h. die Zugangsinformationen bestehen

– entweder aus einer UTM-Benutzerkennung
– oder aus einer UTM-Benutzerkennung und einem Passwort.

← return_code Ergebnis des Funktionsaufrufs

Ergebnis (*return_code*)

CM_OK
Aufruf ok

CM_PROGRAM_STATE_CHECK
Die Conversation ist nicht im Zustand "Initialize".

CM_PROGRAM_PARAMETER_CHECK
: Der Wert in *conversation_ID* ist ungültig oder der Wert in *security_type* ist undefiniert.

CM_PARM_VALUE_NOT_SUPPORTED
: In *security_type* wurde ein von CPI-C nicht unterstützter Wert eingetragen.

CM_PRODUCT_SPECIFIC_ERROR
: Die UPIC-Instanz konnte nicht gefunden werden.

Falls das Ergebnis nicht CM_OK ist, bleibt die Characteristic *security_type* unverändert.

Zustandsänderung

Keine Zustandsänderung.

Hinweis

- Wird in *security_type* der Wert CM_SECURITY_PROGRAM eingetragen, dann müssen Benutzerkennung und ggf. Passwort gesetzt werden mit den Aufrufen *Set_Conversation_Security_User_ID* und *Set_Conversation_Security_Password*.
- Wenn für die Zugangsprüfung nur die Benutzerkennung benötigt wird, ist der Aufruf *Set_Conversation_Security_Password* nicht notwendig.

Verhalten im Fehlerfall

CM_PROGRAM_PARAMETER_CHECK
: Programm ändern.

CM_PARM_VALUE_NOT_SUPPORTED
: Programm ändern.

CM_PRODUCT_SPECIFIC_ERROR
: Das Betriebssystem kann nicht genügend Speicherplatz für interne Puffer bereitstellen. Überprüfen Sie Ihr Programm auf zu hohe Speicherplatzanforderung und starten Sie ggf. Ihr System neu.

Funktionsdeklaration: Set_Conversation_Security_Type

```
CM_ENTRY Set_Conversation_Security_Type (
                           unsigned char CM_PTR  conversation_ID,
          CM_CONVERSATION_SECURITY_TYPE  CM_PTR   conversation_security_type,
                          CM_RETURN_CODE CM_PTR  return_code )
```

Set_Conversation_Security_User_ID - UTM-Benutzerkennung setzen

Die Funktion *Set_Conversation_Security_User_ID* (CMSCSU) setzt die Werte für die Characteristics *security_user_ID* und *security_user_ID_length* der Conversation.
Unter der *security_user_ID* versteht man eine Benutzerkennung einer UTM-Anwendung.

Ein Programm kann eine Benutzerkennung nur dann angeben, wenn die Characteristic *security_type* auf CM_SECURITY_PROGRAM gesetzt ist.

Der Aufruf darf nach *Allocate* nicht mehr ausgeführt werden.

Diese Funktion gehört zu den Advanced Functions.

Syntax

CMSCSU (conversation_ID, security_user_ID, security_user_ID_length, return_code)

Parameter

→ conversation_ID Identifikation der Conversation

→ security_user_ID Benutzerkennung, die zum Einrichten der Conversation benutzt wird. Die UTM-Partner-Anwendung verwendet die Benutzerkennung und ggf. das Passwort, um die Zugangsberechtigung zu überprüfen.

Zusätzlich kann die Partner-Anwendung die Benutzerkennung zur Protokollierung oder zur Abrechnung verwenden.

→ security_user_ID_length
Länge der in *security_user_ID* angegebenen Benutzerkennung in Byte.

Minimum: 0, Maximum: 8

Wird hier 0 angegeben, obwohl *security_type* im Aufruf *Set_Conversation_Security_Type* auf den Wert CM_SECURITY_PROGRAM gesetzt wurde, dann kommt keine Verbindung zu openUTM zustande (Fehler beim Aufruf *Allocate*).

← return_code Ergebnis des Funktionsaufrufs

Ergebnis (*return_code*)

CM_OK
: Aufruf ok

CM_PROGRAM_STATE_CHECK
: Die Conversation ist nicht im Zustand "Initialize" oder *security_type* ist nicht auf CM_SECURITY_PROGRAM gesetzt.

CM_PROGRAM_PARAMETER_CHECK
: Der Wert in *conversation_ID* ist ungültig oder der Wert in *security_user_ID_length* ist kleiner als 0 oder größer als 8.

CM_PRODUCT_SPECIFIC_ERROR
: Die UPIC-Instanz konnte nicht gefunden werden.

Falls das Ergebnis nicht CM_OK ist, bleiben die Characteristics *security_user_ID* und *security_user_ID_length* unverändert.

Zustandsänderung

Keine Zustandsänderung.

Hinweis

- Eine ungültige Benutzerkennung wird bei diesem Aufruf nicht entdeckt. Die Partner-Anwendung überprüft die Benutzerkennung nach dem Einrichten der Conversation auf Gültigkeit. Bei ungültiger Benutzerkennung lehnt der UTM-Server die Conversation ab.
- Das Programm erkennt eine ungültige Benutzerkennung oder ein fehlerhaftes Passwort durch den Returncode CM_SECURITY_NOT_VALID. Dieser wird nach einem dem *Allocate* folgendem *Receive*-Aufruf zurückgegeben.
- Wird im Aufruf *Set_Conversation_Security_Type* der Parameter *security_type* auf CM_SECURITY_NONE gesetzt, dann ist der Aufruf *Set_Conversation_Security_User_ID* nicht erlaubt.

Verhalten im Fehlerfall

CM_PROGRAM_PARAMETER_CHECK
: Programm ändern.

CM_PRODUCT_SPECIFIC_ERROR
: Das Betriebssystem kann nicht genügend Speicherplatz für interne Puffer bereitstellen. Überprüfen Sie Ihr Programm auf zu hohe Speicherplatzanforderung und starten Sie ggf. Ihr System neu.

Funktionsdeklaration: Set_ Conversation_Security_User_ID

```
CM_ENTRY   Set_Conversation_Security_User_ID (
                          unsigned char CM_PTR  conversation_ID,
                          unsigned char CM_PTR  security_user_ID,
                                CM_INT32 CM_PTR  security_user_ID_length,
                         CM_RETURN_CODE CM_PTR  return_code )
```

Set_Convertion – Setzen der Conversation Characteristic CHARACTER_CONVERTION

Der Aufruf *Set_Convertion* (CMSCNV) setzt für die Conversation die Characteristic *CHARACTER_CONVERTION*.

Set_Convertion ändert die Werte, die beim *Initialize_Conversation*-Aufruf aus der Side Information entnommen wurden. Die geänderten Werte gelten nur für die Dauer einer Conversation; die Werte in der Side Information selbst werden nicht verändert.

Der *Set_Convertion*-Aufruf darf nach *Allocate* nicht mehr ausgeführt werden.

Diese Funktion gehört zu den zusätzlichen Funktionen des Trägersystems UPIC; sie ist nicht Bestandteil der CPI-C-Schnittstelle.

Syntax

CMSCNV (conversation_ID, character_convertion, return_code)

Parameter

→ conversation_ID Identifikation der Conversation

→ character_convertion

legt fest, ob eine Code-Konvertierung der Benutzerdaten durchgeführt werden soll oder nicht.

Für *character_convertion* können folgende Werte gesetzt werden:

CM_NO_CHARACTER_CONVERTION
Es findet keine automatische Code-Konvertierung beim Senden oder Empfangen von Daten statt.

CM_IMPLICIT_CHARACTER_CONVERTION
Beim Senden und Empfangen von Daten werden die Daten automatisch konvertiert (siehe auch Abschnitt „Code-Konvertierung“ auf Seite 68).

← return_code Ergebnis des Funktionsaufrufes

Ergebnis (*return_code*)

CM_OK
: Aufruf OK

CM_PROGRAM_PARAMETER_CHECK
: Der Wert in *conversation_ID* oder der Wert für *CHARACTER_CONVERTION* ist ungültig.

CM_PRODUCT_SPECIFIC_ERROR
: Die UPIC-Instanz konnte nicht gefunden werden.

CM_PROGRAM_STATE_CHECK
: Die Conversation ist nicht im Zustand “Initialize”

Zustandsänderung

Der Aufruf ändert den Zustand der Conversation nicht.

Hinweis

Falls der Returncode von CM_OK verschieden ist, bleibt die Characteristic unverändert.

Verhalten im Fehlerfall

CM_PROGRAM_STATE_CHECK
: Programm ändern.

CM_PROGRAM_PARAMETER_CHECK
: Programm ändern.

CM_PRODUCT_SPECIFIC_ERROR
: Das Betriebssystem kann nicht genügend Speicherplatz für interne Puffer bereitstellen. Überprüfen Sie Ihr Programm auf zu hohe Speicherplatzanforderung und starten Sie ggf. Ihr System neu.

Funktionsdeklaration: Set_Convertion

```
CM_ENTRY Set_Convertion(
                  unsigned char                  CM_PTR  conversation_ID,
                  CM_CHARACTER_CONVERTION_TYPE   CM_PTR  convertion_type,
                  CM_RETURN_CODE                 CM_PTR  return_code )
```

Set_Deallocate_Type - Characteristic deallocate_type setzen

Der Aufruf *Set_Deallocate_Type* (CMSDT) setzt den Wert für die Characteristic *deallocate_type* einer Conversation.

Dieser Aufruf gehört zu den Advanced Functions.

Syntax

CMSDT (conversation_ID, deallocate_type, return_code)

Parameter

→ conversation_ID	Identifikation der Conversation
→ deallocate_type	Gibt den Typ für die Beendigung der Conversation an.
	deallocate_type muss den Wert CM_DEALLOCATE_ABEND haben.
← return_code	Ergebnis des Funktionsaufrufs

Ergebnis (*return_code*)

CM_OK
: Aufruf ok

CM_PROGRAM_PARAMETER_CHECK
: Der Wert in *conversation_ID* ist ungültig oder der Wert für *deallocate_type* liegt nicht im zulässigen Wertebereich. Der Wert für *deallocate_type* bleibt unverändert.

CM_PRODUCT_SPECIFIC_ERROR
: Der Wert für *deallocate_type* ist nicht CM_DEALLOCATE_ABEND.
Der Wert für *deallocate_type* bleibt unverändert

Zustandsänderung

Keine Zustandsänderung.

Hinweis

Der *deallocate_type* CM_DEALLOCATE_ABEND wird von einem Programm verwendet, um eine Conversation bedingungslos zu beenden (ohne Berücksichtigung des gegenwärtigen Zustands). Diese abnormale Beendigung sollte vom Programm nur in Ausnahmesituationen durchgeführt werden.

Verhalten im Fehlerfall

CM_PROGRAM_SPECIFIC_ERROR
Programm ändern.

CM_PROGRAM_PARAMETER_CHECK
Programm ändern.

Funktionsdeklaration: Set_Deallocate_Type

```
CM_ENTRY Set_Deallocate_Type ( unsigned char CM_PTR  conversation_ID,
                            CM_DEALLOCATE_TYPE CM_PTR  deallocate_type,
                                CM_RETURN_CODE CM_PTR  return_code )
```

Set_Function_Key - UTM-Funktionstaste setzen

Der Aufruf *Set_Function_Key* (CMSFK) setzt den Wert für die Characteristic *function_key*. *function_key* spezifiziert eine Funktionstaste der UTM-Partner-Anwendung.

Der Wert von *function_key* wird zusammen mit den Daten des nächsten *Send_Data*- bzw. *Send_Mapped_Data*-Aufrufs an die UTM-Anwendung übertragen und die Funktion, die dieser Funktionstaste in der UTM-Anwendung zugeordnet ist, ausgeführt. Das CPI-C-Programm hat dann „die Funktionstaste gedrückt“.

Der Aufruf *Set_Function_Key* ist nur im Zustand "Send" oder "Receive" erlaubt.

Set_Function_Key ist nicht Bestandteil der CPI-C-Spezifikation, sondern eine zusätzliche Funktion des UPIC-Trägersystems.

Syntax

CMSFK (conversation_ID, function_key, return_code)

Parameter

→ conversation_ID Identifikation der Conversation

→ function_key „Funktionstaste“, die das lokale CPI-C-Programm in der fernen UTM-Anwendung „drücken“ will.

Die Funktionstasten sind in der Form CM_FKEY_*ftaste* anzugeben. Dabei ist für *ftaste* die Nummer der K- bzw. F-Taste anzugeben, die „gedrückt“ werden soll.
Beispiel: Soll Funktionstaste F10 der UTM-Partner-Anwendung „gedrückt“ werden, dann geben Sie für *function_key* den Wert CM_FKEY_F10 an.

openUTM auf Unix- und Windows-Systemen unterstützt die Funktionstasten F1 bis F20.
openUTM auf BS2000/OSD unterstützt die Funktionstasten K1 bis K14 und F1 bis F24.

Der Wert CM_UNMARKED bedeutet, dass keine Funktionstaste gesetzt wird.

← return_code Ergebnis des Funktionsaufrufs

Ergebnis (*return_code*)

CM_OK
: Aufruf ok

CM_PROGRAM_STATE_CHECK
: Die Conversation ist nicht im Zustand "Send" oder "Receive".

CM_PROGRAM_PARAMETER_CHECK
: Der Wert in *conversation_ID* oder der Wert in *function_key* ist ungültig.

CM_PRODUCT_SPECIFIC_ERROR
: Die UPIC-Instanz konnte nicht gefunden werden.

CM_MAP_ROUTINE_ERROR
: In der UTM-Partner-Anwendung werden keine Funktionstasten im UPIC-Protokoll unterstützt.

Zustandsänderung

Im fehlerfreien Fall liefert die Funktion das Ergebnis CM_OK zurück. Dieser Aufruf ändert den Zustand des Programms nicht.

Hinweis

- Bei openUTM auf Unix- und Windows-Systemen haben Funktionstasten nur im Formatmodus eine Wirkung, d.h. wenn zum Austausch der Daten die Aufrufe *Send_Mapped_Data* und *Receive_Mapped_Data* verwendet werden.
- Die in *Set_Function_Key* spezifizierte Funktionstaste wird erst zusammen mit den Daten des folgenden *Send_Data-* bzw. *Send_Mapped_Data-*Aufrufs an die UTM-Partner-Anwendung übergeben.
  Sobald der Wert von *function_key* an openUTM gesendet wird, wird *function_key* im lokalen CPI-C-Programm auf CM_UNMARKED (keine Funktionstaste) zurückgesetzt.
- Empfängt die UTM-Partner-Anwendung von einem UPIC-Client eine Funktionstaste, so wird nur der Parameter RET der Steueranweisung SFUNC, die die Funktionstaste beschreibt, ausgewertet. RET enthält den Returncode, der nach dem MGET-Aufruf des UTM-Vorgangs im Feld KCRCCC des Kommunikationsbereichs steht. Ist der Parameter RET für die Funktionstaste nicht generiert, dann liefert openUTM beim MGET-Aufruf immer den Returncode 19Z (Funktionstaste nicht generiert oder Sonderfunktion ungültig).

Verhalten im Fehlerfall

CM_PROGRAM_STATE_CHECK
: Programm ändern.

CM_PROGRAM_PARAMETER_CHECK
: Programm ändern.

CM_PRODUCT_SPECIFIC_ERROR
: Das Betriebssystem kann nicht genügend Speicherplatz für interne Puffer bereitstellen. Überprüfen Sie Ihr Programm auf zu hohe Speicherplatzanforderung und starten Sie ggf. Ihr System neu.

Funktionsdeklaration: Set_Function_Key

```
CM_ENTRY Set_Function_Key ( unsigned char CM_PTR  conversation_ID,
                                   CM_INT32 CM_PTR  function_key,
                             CM_RETURN_CODE CM_PTR  return_code)
```

Set_Partner_Host_Name - Hostname der Partner-Anwendung setzen

Der Aufruf *Set_Partner_Host_Name* (CMSPHN) setzt den Wert für die Characteristic *HOSTNAME* der Partner-Anwendung der Conversation. Der Aufruf überschreibt den Wert, der beim *Initialize_Conversation*-Aufruf zugewiesen wurde. Er darf nach dem Aufruf *Allocate* nicht mehr aufgerufen werden.

Diese Funktion gehört zu den zusätzlichen Funktionen des Trägersystems UPIC; sie ist nicht Bestandteil der CPI-C-Schnittstelle.

X/W *UPIC-Local:*



Der Aufruf Set_Partner_Host_Name wird bei der Anbindung über UPIC-L nicht unterstützt.

UPIC-R mit openUTM-Cluster-Nutzung:

Der Aufruf *Set_Partner_Host_Name* wird nicht unterstützt, wenn ein openUTM-Cluster konfiguriert ist.

Syntax

CMSPHN (conversation_ID, host_name, host_name_length, return_code)

Parameter

→ conversation_ID	Identifikation der Conversation
→ host_name	legt fest, welcher Hostname verwendet wird
→ host_name_length	legt die Länge des *host_name* in Byte fest. Minimum:1, Maximum:32
← return_code	Ergebnis des Funktionsaufrufs

Ergebnis (*return_code*)

CM_OK
: Aufruf ok

CM_CALL_NOT_SUPPORTED
: Die Funktion wird nicht unterstützt.



Bei UPIC-L tritt der Returncode immer auf. Er zeigt dem Programm an, dass kein *host_name* verwendet werden kann, da UPIC-L diese Information auf Grund des darunterliegenden Kommunikationssystems nicht benötigt.

Bei UPIC-R tritt der Returncode nur auf, wenn ein openUTM-Cluster konfiguriert wurde. Er zeigt dem Programm an, dass *host_name* nicht geändert werden kann.

CM_PROGRAM_PARAMETER_CHECK
Der Wert der *conversation_ID* oder für *host_name_length* ist ungültig.

CM_PROGRAM_STATE_CHECK
Die Conversation ist nicht im Zustand "Init".

CM_PRODUCT_SPECIFIC_ERROR
Die UPIC-Instanz konnte nicht gefunden werden.

Zustandsänderung

Der Aufruf ändert den Zustand der Conversation nicht.

Hinweis

Der Wert von *host_name* wird ignoriert, wenn auch für *ip_adress* ein Wert gesetzt ist, entweder in der `upicfile` oder durch einen *Set_Partner_IP_Adress*-Aufruf im UPIC-Programm.

Verhalten im Fehlerfall

CM_PROGRAM_PARAMETER_CHECK
Programm ändern.

CM_PROGRAM_STATE_CHECK
Programm ändern.

CM_PRODUCT_SPECIFIC_ERROR
Das Betriebssystem kann nicht genügend Speicherplatz für interne Puffer bereitstellen. Überprüfen Sie Ihr Programm auf zu hohe Speicherplatzanforderung und starten Sie ggf. Ihr System neu.

CM_CALL_NOT_SUPPORTED
Muss kein Fehler sein: Das Programm kann sich diesen Returncode merken und auf weitere Aufrufe zum Setzen von Adressinformationen verzichten.



Funktionsdeklaration: Set_Partner_Host_Name

```
CM_ENTRY Set_Partner_Host_Name( unsigned char CM_PTR  conversation_ID,
                                unsigned char CM_PTR  host_name,
                                     CM_INT32 CM_PTR  host_name_lth,
                               CM_RETURN_CODE CM_PTR  return_code )
```

Set_Partner_IP_Address - IP-Adresse der Partner-Anwendung setzen

Der Aufruf *Set_Partner_IP_Address* (CMSPIA) setzt den Wert für die Characteristic *IP-ADDRESS* der Partner-Anwendung der Conversation. Der Aufruf überschreibt den Wert, der beim *Initialize_Conversation-Aufruf* zugewiesen wurde. Er darf nach dem Aufruf *Allocate* nicht mehr aufgerufen werden.

Diese Funktion gehört zu den zusätzlichen Funktionen des Trägersystems UPIC; sie ist nicht Bestandteil der CPI-C-Schnittstelle.

X/W *UPIC-Local:*


Der Aufruf Set_Partner_IP_Address wird bei der Anbindung über UPIC-L nicht unterstützt.

UPIC-R mit openUTM-Cluster-Nutzung:

Der Aufruf *Set_Partner_IP_Address* wird nicht unterstützt, wenn ein openUTM-Cluster konfiguriert ist.

Syntax

CMSPIA (conversation_ID, ip_address, ip_address_length, return_code)

Parameter

→ conversation_ID	Identifikation der Conversation
→ ip_address	legt fest, dass statt der Characteristic *hostname* eine IP-Adresse verwendet wird.
→ ip_address_length	legt die Länge von *ip_address* in Byte fest. Minimum:0, Maximum:64.
← return_code	Ergebnis des Funktionsaufrufs

Ergebnis (*return_code*)

CM_OK
Aufruf ok

CM_CALL_NOT_SUPPORTED
Die Funktion wird nicht unterstützt.

X/W X/W X/W Bei UPIC-L tritt dieser Returncode immer auf. Er zeigt dem Programm an, dass keine *ip_address* verwendet werden kann, da UPIC-L diese Information aufgrund des darunterliegenden Kommunikationssystems nicht benötigt.

Bei UPIC-R tritt der Returncode auf, wenn ein openUTM-Cluster konfiguriert ist. Er zeigt dem Programm an, dass die *ip_address* nicht geändert werden kann.

B B B B Bei UPIC-R für BS2000/OSD tritt der Returncode auf, wenn die UPIC-Bibliothek auf BS2000 zusammen mit CMX eingesetzt wird. Das von UPIC-R verwendete Kommunikationssystem CMX bietet auf BS2000-Systemen keine Möglichkeit, an der Schnittstelle IP-Adressen zur Adressierung der Partner-Anwendung zu übergeben.

CM_PROGRAM_PARAMETER_CHECK
Der Wert der *conversation_ID* oder für *ip_address_length* ist ungültig.

CM_PROGRAM_STATE_CHECK
Die Conversation ist nicht im Zustand "Init".

CM_PRODUCT_SPECIFIC_ERROR
Die UPIC-Instanz konnte nicht gefunden werden.

Zustandsänderung

Der Aufruf ändert den Zustand der Conversation nicht.

Hinweis

- *ip_address* wird für IPv4 in der üblichen Punktnotation angegeben:

  ```
  xxx.xxx.xxx.xxx
  ```

  Die einzelnen Oktette xxx sind auf 3 Stellen beschränkt. Der Inhalt der Oktette wird immer als Dezimalzahl interpretiert. Insbesondere bedeutet dies, dass Oktette, die links mit Nullen aufgefüllt sind, **nicht** als Oktalzahl interpretiert werden.

- *ip_address* wird für IPv6 in der üblichen Doppelpunktnotation angegeben:

  `x:x:x:x:x:x:x:x`

  `x` ist eine Hexadezimalzahl zwischen 0 und FFFF. Die alternativen Schreibweisen für IPv6-Adressen sind erlaubt (vgl. RFC2373).

  Wenn in der IPv6 Adresse eine embedded IPv4 Adresse in Punktnotation angegeben ist, dann gilt für die Oktette der IPv4 Adresse das gleiche wie oben. Die Oktette werden immer als Oktalzahl interpretiert.

- Wenn *ip_adress* und HOST_NAME gesetzt sind, wird der Wert von HOST_NAME ignoriert.

Verhalten im Fehlerfall

CM_PROGRAM_PARAMETER_CHECK
: Programm ändern.

CM_PROGRAM_STATE_CHECK
: Programm ändern.

CM_PRODUCT_SPECIFIC_ERROR
: Das Betriebssystem kann nicht genügend Speicherplatz für interne Puffer bereitstellen. Überprüfen Sie Ihr Programm auf zu hohe Speicherplatzanforderung und starten Sie ggf. Ihr System neu.

CM_CALL_NOT_SUPPORTED
: Muss kein Fehler sein: Das Programm kann sich diesen Returncode merken und auf weitere Aufrufe zum Setzen von Adressinformationen verzichten.

Funktionsdeklaration: Set_Partner_IP_Address

```
CM_Entry Set_Partner_IP_Address ( unsigned char CM_PTR  conversation_ID,
                                  unsigned char CM_PTR  ip_address,
                                       CM_INT32 CM_PTR  ip_address_length,
                                 CM_RETURN_CODE CM_PTR  return_code )
```

Set_Partner_LU_Name - Setzen der Conversation Characteristics partner_LU_name

Der Aufruf *Set_Partner_LU_Name* (CMSPLN) setzt für die Conversation die Characteristics *partner_LU_name* und *partner_LU_name_length*.

Set_Partner_LU_Name ändert die Werte, die beim *Initialize_Conversation*-Aufruf aus der Side Information entnommen wurden. Die geänderten Werte gelten nur für die Dauer einer Conversation; die Werte in der Side Information selbst werden nicht verändert.

Der *Set_Partner_LU_Name*-Aufruf darf nach *Allocate* nicht mehr ausgeführt werden.

Dieser Aufruf gehört zu den Advanced Functions.

UPIC-R mit openUTM-Cluster-Nutzung:

Der Aufruf *Set_Partner_LU_Name* wird nicht unterstützt, wenn ein openUTM-Cluster konfiguriert ist.

Syntax

CMSPLN (conversation_ID, partner_LU_name, partner_LU_name_length, return_code)

Parameter

→ conversation_ID Identifikation der Conversation.

→ partner_LU_name Legt fest, welcher *partner_LU_name* verwendet werden soll.

→ partner_LU_name_length
Gibt die Länge von *partner_LU_name* an.
Minimum: 1, Maximum: 32.
UPIC-L:
Minimum: 1, Maximum: 8.

← return_code Ergebnis des Funktionsaufrufs.

Ergebnis (*return_code*)

CM_OK
Aufruf ok

CM_PROGRAM_PARAMETER_CHECK
Der Wert der *conversation_ID* ist ungültig oder *partner_LU_name* ist ungültig oder der Wert in *partner_LU_name_length* ist kleiner als 1 oder größer als 32.

CM_PROGRAM_STATE_CHECK
Die Conversation ist nicht im Zustand "Initialize".

CM_PRODUCT_SPECIFIC_ERROR
Die UPIC-Instanz konnte nicht gefunden werden.

CM_CALL_NOT_SUPPORTED
Die Funktion wird nicht unterstützt.

Der Returncode tritt bei UPIC-R auf, wenn ein openUTM-Cluster konfiguriert ist. Er zeigt dem Programm an, dass der *partner_LU_name* nicht geändert werden kann.

Zustandsänderung

Der Aufruf ändert den Zustand der Conversation nicht.

Hinweis

- Falls der Returncode von CM_OK verschieden ist, bleibt die Characteristic *partner_LU_name* unverändert.
- Mit diesem Aufruf wird lediglich die Characteristic *partner_LU_name* gesetzt. Ein ungültiger *partner_LU_name* wird bei diesem Aufruf nicht entdeckt. Erst der *Allocate*-Aufruf erkennt einen ungültigen *partner_LU_name*, wenn er keine Transportverbindung zur UTM-Anwendung aufbauen kann. Er liefert dann den return_code CM_ALLOCATE_FAILURE_NO_RETRY zurück.
- Falls eine Anwendung mit UPIC-L gebunden ist und einen *partner_LU_name* mit einer Länge > 8 übergibt, so liefert der Aufruf *Set_Partner_LU_Name* den Returncode CM_OK. Im nachfolgenden *Allocate*-Aufruf wird der *partner_LU_name* aber stillschweigend nach 8 Byte abgeschnitten.

Verhalten im Fehlerfall

CM_PROGRAM_STATE_CHECK
Programm ändern.

CM_PROGRAM_PARAMETER_CHECK
Programm ändern.

CM_PRODUCT_SPECIFIC_ERROR
Das Betriebssystem kann nicht genügend Speicherplatz für interne Puffer bereitstellen. Überprüfen Sie Ihr Programm auf zu hohe Speicherplatzanforderung und starten Sie ggf. Ihr System neu.

CM_CALL_NOT_SUPPORTED
Muss kein Fehler sein: Das Programm kann sich diesen Returncode merken und auf weitere Aufrufe zum Setzen von Adressinformationen verzichten.

Funktionsdeklaration: Set_Partner_LU_Name

```
CM_ENTRY Set_Partner_LU_Name ( unsigned char CM_PTR  conversation_ID,
                               unsigned char CM_PTR  partner_LU_name,
                                    CM_INT32 CM_PTR  partne_LU_name_length,
                              CM_RETURN_CODE CM_PTR  return_code )
```

Set_Partner_Port - TCP/IP-Port der Partner-Anwendung setzen

Der Aufruf *Set_Partner_Port* (CMSPP) setzt die Portnummer für TCP/IP für die Partner-Anwendung und damit die Conversation Characteristic *PORT*. Der Aufruf überschreibt den Wert, der beim *Initialize_Conversation*-Aufruf zugewiesen wurde. Er darf nach dem Aufruf *Allocate* nicht mehr aufgerufen werden.

Diese Funktion gehört zu den zusätzlichen Funktionen des Trägersystems UPIC; sie ist nicht Bestandteil der CPI-C-Schnittstelle.

UPIC-Local:

Der Aufruf Set_Partner_Port wird bei der Anbindung über UPIC-L nicht unterstützt.

Syntax

CMSPP (conversation_ID, listener_port, return_code)

Parameter

→ conversation_ID Identifikation der Conversation

→ port_number legt fest, welche Portnummer der Partner-Anwendung beim Kommunikationssystem gesucht wird.
Minimum: 0; Maximum: 32767

← return_code Ergebnis des Funktionsaufrufs

Ergebnis (*return_code*)

CM_OK
Aufruf ok

CM_CALL_NOT_SUPPORTED
Die Funktion wird nicht unterstützt. Dieser Returncode tritt bei UPIC-L und bei UPIC-R auf BS2000/OSD auf.

X/W X/W X/W Bei UPIC-L tritt dieser Returncode immer auf. Er zeigt dem Programm an, dass keine Portnummer vergeben werden kann, da UPIC-L diese Information auf Grund des darunterliegenden Kommunikationssystems nicht benötigt.

B B B B B B Bei UPIC-R (BS2000) tritt der Returncode nur auf, wenn die UPIC Bibliothek auf BS2000/OSD zusammen mit CMX eingesetzt wird. Das von UPIC-R verwendete Kommunikationssystem CMX bietet auf BS2000-Systemen keine Möglichkeit, an der Schnittstelle IP-Adressen zur Adressierung der Partner-Anwendung zu übergeben. Wenn die UPIC-Bibliothek die Socketschnittstelle als Kommunikationssystem verwendet, dann tritt der Returncode nie auf.

CM_PROGRAM_PARAMETER_CHECK
: Der Wert der *conversation_ID* oder der *port_number* ist ungültig.

CM_PROGRAM_STATE_CHECK
: Die Conversation ist nicht im Zustand "Init".

CM_PRODUCT_SPECIFIC_ERROR
: Die UPIC-Instanz konnte nicht gefunden werden.

Zustandsänderung

Der Aufruf ändert den Zustand der Conversation nicht.

Verhalten im Fehlerfall

CM_PROGRAM_PARAMETER_CHECK
: Programm ändern.

CM_PROGRAM_STATE_CHECK
: Programm ändern.

CM_PRODUCT_SPECIFIC_ERROR
: Das Betriebssystem kann nicht genügend Speicherplatz für interne Puffer bereitstellen. Überprüfen Sie Ihr Programm auf zu hohe Speicherplatzanforderung und starten Sie ggf. Ihr System neu.

CM_CALL_NOT_SUPPORTED

X/W Muss kein Fehler sein: Falls eine Anwendung sowohl für UPIC-L als auch für UPIC-R vorgesehen ist, bedeutet dieser Returncode auf Unix- und Windows-Systemen lediglich, dass die Anwendung mit einer UPIC-L-Bibliothek gebunden ist. Das Programm kann sich diesen Returncode merken und auf weitere Aufrufe zum Setzen von Adressinformationen verzichten.

B Auf BS2000/OSD bedeutet dieser Returncode, dass die Anwendung mit UPIC-R und CMX gebunden ist. Das Programm kann sich diesen Returncode merken und auf die Aufrufe *Set_Partner_IP_Address* und *Set_Partner_Port* verzichten.

Funktionsdeklaration: Set_Partner_Port

```
CM_ENTRY Set_Partner_Port ( unsigned char CM_PTR  conversation_ID,
                                   CM_INT32 CM_PTR  port_number,
                             CM_RETURN_CODE CM_PTR  return_code )
```

Set_Partner_Tsel - T-SEL der Partner-Anwendung setzen

Der Aufruf *Set_Partner_Tsel* (CMSPT) setzt den Wert für die Characteristic *T-SEL* der Partner-Anwendung der Conversation. Der Aufruf überschreibt den Wert, der beim *Initialize_Conversation*-Aufruf zugewiesen wurde. Er darf nach dem Aufruf *Allocate* nicht mehr aufgerufen werden.

Diese Funktion gehört zu den zusätzlichen Funktionen des Trägersystems UPIC; sie ist nicht Bestandteil der CPI-C-Schnittstelle.



X/W *UPIC-Local:*

X/W Der Aufruf *Set_Partner_Tsel* wird bei der Anbindung über UPIC-L nicht unterstützt.

Syntax

CMSPT (conversation_ID, transport_selector, transport_selector_length, return_code)

Parameter

→ conversation_ID Identifikation der Conversation

→ transport_selector Transport-Selektor der Partner-Anwendung, der dem Kommunikationssystem übergeben wird.

→ transport_selector_length
Länge des Transport-Selektors in Byte.
Minimum: 0, Maximum: 8

Wird die Länge des Transport-Selektors mit 0 angegeben, so wird der erste Namensteil des *partner_LU_name* als Transport-Selektor verwendet.

← return_code Ergebnis des Funktionsaufrufs

Ergebnis (*return_code*)

CM_OK
Aufruf ok



CM_CALL_NOT_SUPPORTED
Die Funktion wird nicht unterstützt. Dieser Returncode tritt nur bei UPIC-L auf. Er zeigt dem Programm an, dass kein TSEL vergeben werden kann, da UPIC-L diese Information auf Grund des darunterliegenden Kommunikationssystems nicht benötigt.

CM_PROGRAM_PARAMETER_CHECK
Der Wert der *conversation_ID* oder der *transport_selector_length* ist ungültig.

CM_PROGRAM_STATE_CHECK
Die Conversation ist nicht im Zustand "Init".

CM_PRODUCT_SPECIFIC_ERROR
Die UPIC-Instanz konnte nicht gefunden werden.

Zustandsänderung

Der Aufruf ändert den Zustand der Conversation nicht.

Verhalten im Fehlerfall

CM_PROGRAM_PARAMETER_CHECK
Programm ändern.

CM_PROGRAM_STATE_CHECK
Programm ändern.

CM_PRODUCT_SPECIFIC_ERROR
Das Betriebssystem kann nicht genügend Speicherplatz für interne Puffer bereitstellen. Überprüfen Sie Ihr Programm auf zu hohe Speicherplatzanforderung und starten Sie ggf. Ihr System neu.

X/W CM_CALL_NOT_SUPPORTED
X/W Muss kein Fehler sein: Falls eine Anwendung sowohl für UPIC-L als auch für
X/W UPIC-R vorgesehen ist, bedeutet dieser Returncode lediglich, dass die Anwendung
X/W mit einer UPIC-L-Bibliothek gebunden ist. Das Programm kann sich diesen Return-
X/W code merken und auf weitere Aufrufe zum Setzen von Adressinformationen verzich-
X/W ten.

Funktionsdeklaration: Set_Partner_Tsel

```
CM_ENTRY Set_Partner_TSEL ( unsigned char CM_PTR  conversation_ID,
                            unsigned char CM_PTR  transport_selector,
                                 CM_INT32 CM_PTR  transport_selector_length,
                           CM_RETURN_CODE CM_PTR  return_code )
```

Set_Partner_Tsel_Format - T-SEL-Format der Partner-Anwendung setzen

Der Aufruf *Set_Partner_Tsel_Format* (CMSPTF) setzt den Wert für die Characteristic *T-SEL-FORMAT* der Partner-Anwendung der Conversation. Der Aufruf überschreibt den Wert, der beim *Initialize_Conversation*-Aufruf zugewiesen wurde. Er darf nach dem Aufruf *Allocate* nicht mehr aufgerufen werden.

Diese Funktion gehört zu den zusätzlichen Funktionen des Trägersystems UPIC; sie ist nicht Bestandteil der CPI-C-Schnittstelle.



UPIC-Local:



Der Aufruf *Set_Partner_Tsel_Format* wird bei der Anbindung über UPIC-L nicht unterstützt.

Syntax

CMSPTF (conversation_ID, tsel_format, return_code)

Parameter

→ conversation_ID Identifikation der Conversation

→ tsel_format legt fest, welcher Zeichensatz für den Transport-Selektor (TSEL) verwendet wird. Folgende Werte können Sie angeben:

– CM_TRANSDATA _FORMAT
  Der Transport-Selektor wird im TRANSDATA-Format an das Kommunikationssystem übergeben.

– CM_EBCDIC_FORMAT
  Der Transport-Selektor wird im EBCDIC Format an das Kommunikationssystem übergeben

– CM_ASCII_FORMAT
  Der Transport-Selektor wird im ASCII-Format an das Kommunikationssystem übergeben.

← return_code Ergebnis des Funktionsaufrufs

Ergebnis (*return_code*)

CM_OK
: Aufruf ok

CM_CALL_NOT_SUPPORTED
: Die Funktion wird nicht unterstützt. Dieser Returncode tritt nur bei UPIC-L auf. Er zeigt dem Programm an, dass kein TSEL-Format vergeben werden kann, da UPIC-L diese Information auf Grund des darunterliegenden Kommunikationssystems nicht benötigt.

CM_PROGRAM_PARAMETER_CHECK
: Der Wert der *conversation_ID* oder von *tsel_format* ist ungültig.

CM_PROGRAM_STATE_CHECK
: Die Conversation ist nicht im Zustand "Init".

CM_PRODUCT_SPECIFIC_ERROR
: Die UPIC-Instanz konnte nicht gefunden werden.

Zustandsänderung

Der Aufruf ändert den Zustand der Conversation nicht.

Verhalten im Fehlerfall

CM_PROGRAM_PARAMETER_CHECK
: Programm ändern.

CM_PROGRAM_STATE_CHECK
: Programm ändern.

CM_PRODUCT_SPECIFIC_ERROR
: Das Betriebssystem kann nicht genügend Speicherplatz für interne Puffer bereitstellen. Überprüfen Sie Ihr Programm auf zu hohe Speicherplatzanforderung und starten Sie ggf. Ihr System neu.



CM_CALL_NOT_SUPPORTED
: Muss kein Fehler sein: Falls eine Anwendung sowohl für UPIC-L als auch für UPIC-R vorgesehen ist, bedeutet dieser Returncode lediglich, dass die Anwendung mit einer UPIC-L-Bibliothek gebunden ist. Das Programm kann sich diesen Returncode merken und auf weitere Aufrufe zum Setzen von Adressinformationen verzichten.

Funktionsdeklaration: Set_Partner_TSEL_Format

```
CM_ENTRY Set_Partner_TSEL_Format ( unsigned char CM_PTR  conversation_ID,
                                   CM_TSEL_Format CM_PTR  tsel_format,
                                   CM_RETURN_CODE CM_PTR  return_code )
```

Set_Receive_Timer - Timer für den blockierenden Receive setzen

Der Aufruf *Set_Receive_Timer* (CMSRCT) setzt den Timeout-Timer für einen blockierenden *Receive*- bzw. *Receive_Mapped_Data*-Aufruf.

Wenn dieser Timer gesetzt ist und für den Datenempfang *receive_type*=CM_RECEIVE_AND_WAIT gesetzt ist, werden die Aufrufe *Receive* und *Receive_Mapped_Data* nach der im Feld *receive_timer* festgelegten Zeit abgebrochen.

Set_Receive_Timer darf nach dem *Allocate*-Aufruf zu jedem beliebigen Zeitpunkt und beliebig oft innerhalb einer Conversation aufgerufen werden. Es gilt jeweils die Timer-Einstellung des letzten *Set_Receive_Timer*-Aufrufs.

Diese Funktion ist nicht Bestandteil der CPI-C Spezifikation, sondern eine zusätzliche Funktion des UPIC-Trägersystems.



UPIC-Local:



Der Aufruf *Set_Receive_Timer* wird bei der Anbindung über UPIC-L nicht unterstützt.

Syntax

CMSRCT (conversation_ID, receive_timer, return_code)

Parameter

→ conversation_ID — Identifikation der Conversation

→ receive_timer — Zeit in Millisekunden, nach der ein blockierender *Receive*- bzw. *Receive_Mapped_Data*-Aufruf unterbrochen wird. Die Aufrufe *Receive*- und *Receive_Mapped_Data* wirken blockierend, wenn die Characteristic *receive_type* den Wert CM_RECEIVE_AND_WAIT hat.
Der Receive-Timer wird zurückgesetzt, wenn Sie *receive_timer* auf 0 setzen. Die Wartezeit des *Receive*- oder *Receive_Mapped_Data*-Aufrufs wird dann nicht mehr überwacht.

Der für *receive_timer* angegebene Wert wird auf die nächste volle Sekunde aufgerundet.

← return_code — Ergebnis des Funktionsaufrufs

Ergebnis (*return_code*)

CM_OK
: Aufruf ok

CM_PROGRAM_STATE_CHECK
: Die Conversation ist nicht im Zustand "Send" oder "Receive".

CM_PROGRAM_PARAMETER_CHECK
: Der Wert in *conversation_ID* ist ungültig oder in *receive_timer* wurde ein Wert < 0 angegeben.

CM_PRODUCT_SPECIFIC_ERROR
: Die UPIC-Instanz konnte nicht gefunden werden.

CM_CALL_NOT_SUPPORTED
: Die Funktion wird nicht unterstützt.

Zustandsänderung

Im fehlerfreien Fall liefert die Funktion das Ergebnis CM_OK zurück. Dieser Aufruf ändert den Zustand der Conversation nicht.

Hinweis

- Der *Set_Receive_Timer* ist nur sinnvoll im Zusammenhang mit den Aufrufen *Receive* und *Receive_Mapped_Data*.
- *Set_Receive_Timer* kann innerhalb einer Conversation beliebig oft aufgerufen werden. Es gilt immer der Wert, der beim letzten Aufruf von *Set_Receive_Timer* vor einem *Receive*- bzw. *Receive_Mapped_Data*-Aufruf gesetzt wurde. Der gesetzte Wert bleibt bis zum nächsten *Set_Receive_Timer*-Aufruf bzw. bis zum Ende der Conversation gültig.

Verhalten im Fehlerfall

CM_PROGRAM_STATE_CHECK
Programm ändern.

CM_PROGRAM_PARAMETER_CHECK
Programm ändern.

CM_PRODUCT_SPECIFIC_ERROR
Das Betriebssystem kann nicht genügend Speicherplatz für interne Puffer bereitstellen. Überprüfen Sie Ihr Programm auf zu hohe Speicherplatzanforderung und starten Sie ggf. Ihr System neu.

X/W CM_CALL_NOT_SUPPORTED
X/W Muss kein Fehler sein: Falls eine Anwendung sowohl für UPIC-L als auch für
X/W UPIC-R vorgesehen ist, bedeutet dieser Returncode lediglich, dass die Anwendung
X/W mit einer UPIC-L-Bibliothek gebunden ist. Das Programm kann sich diesen Return-
X/W code merken und auf weitere *Set_Receive_Timer* Aufrufe verzichten.

Funktionsdeklaration: Set_Receive_Timer

```
CM_ENTRY Set_Receive_Timer ( unsigned char CM_PTR  conversation_ID,
                                 CM_TIMEOUT CM_PTR  timeout_time,
                             CM_RETURN_CODE CM_PTR  return_code )
```

Set_Receive_Type - Empfangsmodus (receive_type) setzen

Der Aufruf *Set_Receive_Type* (CMSRT) setzt den Wert für die Conversation Characteristic *receive_type*. In *receive_type* legen Sie fest, ob die *Receive*- und *Receive_Mapped_Data*-Aufrufe blockierend oder nicht-blockierend ausgeführt werden. Der Aufruf überschreibt den Wert von *receive_type*, der beim *Initialize_Conversation*-Aufruf zugewiesen wurde.

Der Aufruf *Set_Receive_Type* ist im Zustand "Initialize", "Send" oder "Receive" erlaubt.

Diese Funktion gehört zu den Advanced Functions.

X/W *UPIC-Local:*


Der Aufruf *Set_Receive_Type* wird bei der Anbindung über UPIC-L nicht unterstützt.

Syntax

CMSRT (conversation_ID, receive_type, return_code)

Parameter

→ conversation_ID Identifikation der Conversation

→ receive_type legt fest, ob die folgenden *Receive-* / *Receive_Mapped_Data*-Aufrufe blockierend oder nicht-blockierend ausgeführt werden. Folgende Werte können Sie angeben:

– CM_RECEIVE_AND_WAIT
Die Aufrufe *Receive* und *Receive_Mapped_Data* wirken blockierend, d.h. liegt zum Aufrufzeitpunkt keine Information vor, wird so lange gewartet, bis Informationen für diese Conversation vorliegen. Erst dann kehrt der Programmlauf aus dem *Receive-* bzw. *Receive_Mapped_Data*-Aufruf zurück und übergibt die Daten an das Programm.
Liegt zum Aufrufzeitpunkt bereits eine Information vor, dann empfängt das Programm sie ohne zu warten.
Wurde vor dem *Receive-* bzw. *Receive_Mapped_Data*-Aufruf mit *Set_Receive_Timer* eine maximale Wartezeit (Timeout-Timer) gesetzt, dann kehrt der Programmlauf nach Ablauf dieser Wartezeit aus dem *Receive-* bzw. *Receive_Mapped_Data*-Aufruf zurück, auch wenn noch keine Information vorliegt.

– CM_RECEIVE_IMMEDIATE
Die Aufrufe *Receive* und *Receive_Mapped_Data* wirken nicht-blockierend, d.h. liegen zum Aufrufzeitpunkt Informationen vor, dann empfängt das Programm sie ohne zu warten.
Liegen zum Aufrufzeitpunkt keine Informationen vor, dann wartet das Programm nicht. Der Programmlauf kehrt sofort aus dem *Receive-* bzw. *Receive_Mapped_Data*-Aufruf zurück.

← return_code Ergebnis des Funktionsaufrufs

Ergebnis (*return_code*)

CM_OK
Aufruf ok

CM_PROGRAM_PARAMETER_CHECK
Der Wert in *conversation_ID* ist ungültig oder der Wert von *receive_type* ist undefiniert.

CM_PRODUCT_SPECIFIC_ERROR
Die UPIC-Instanz konnte nicht gefunden werden.

CM_CALL_NOT_SUPPORTED
Die Funktion wird nicht unterstützt.

Zustandsänderung

Im fehlerfreien Fall liefert die Funktion das Ergebnis CM_OK zurück. Dieser Aufruf ändert den Zustand der Conversation nicht.

Hinweis

- Falls der Returncode von CM_OK verschieden ist, bleibt die Characteristic *receive_type* unverändert.
- Wird *Set_Receive_Type* im Zustand "Start" oder "Reset" aufgerufen, dann ist der in *conversation_ID* übergebene Wert immer ungültig. Als Ergebnis des Aufrufs wird dann immer der Returncode CM_PROGRAM_PARAMETER_CHECK zurückgeliefert.

Verhalten im Fehlerfall

CM_PROGRAM_PARAMETER_CHECK
: Programm ändern.

CM_PRODUCT_SPECIFIC_ERROR
: Das Betriebssystem kann nicht genügend Speicherplatz für interne Puffer bereitstellen. Überprüfen Sie Ihr Programm auf zu hohe Speicherplatzanforderung und starten Sie ggf. Ihr System neu.

X/W CM_CALL_NOT_SUPPORTED
X/W Muss kein Fehler sein: Falls eine Anwendung sowohl für UPIC-L als auch für
X/W UPIC-R vorgesehen ist, bedeutet dieser Returncode lediglich, dass die Anwendung
X/W mit einer UPIC-L-Bibliothek gebunden ist. Das Programm kann sich diesen Return-
X/W code merken und auf weitere Set_Receive_Type Aufrufe verzichten.

Funktionsdeklaration: Set_Receive_Type

```
CM_ENTRY Set_Receive_Type ( unsigned char CM_PTR  conversation_ID,
                            CM_RECEIVE_TYPE CM_PTR  receive_type,
                             CM_RETURN_CODE CM_PTR  return_code )
```

Set_Sync_Level - Synchronisationsstufe (sync_level) setzen

Der Aufruf *Set_Sync_Level* (CMSSL) setzt den Wert für die Characteristic *sync_level* einer Conversation. Der Aufruf überschreibt den Wert, der beim *Initialize_Conversation*-Aufruf zugewiesen wurde.

Der *Set_Sync_Level*-Aufruf darf nach dem *Allocate*-Aufruf nicht mehr ausgeführt werden.

Diese Funktion gehört zu den Advanced Functions.

Syntax

CMSSL (conversation_ID, sync_level, return_code)

Parameter

→ conversation_ID	Identifikation der Conversation
→ sync_level	gibt die Stufe der Synchronisation an, die das lokale CPI-C-Programm und die entfernteUTM-Anwendung über diese Conversation benutzen können. *sync_level* muss den Wert CM_NONE haben.
← return_code	Ergebnis des Funktionsaufrufs

Ergebnis (*return_code*)

CM_OK
    Aufruf ok

CM_PROGRAM_STATE_CHECK
    Die Conversation ist nicht im Zustand "Initialize".

CM_PROGRAM_PARAMETER_CHECK
    Der Wert in *conversation_ID* ist ungültig oder der Wert in *sync_level* ist undefiniert.

CM_PRODUCT_SPECIFIC_ERROR
    Die UPIC-Instanz konnte nicht gefunden werden.

Zustandsänderung

Im fehlerfreien Fall liefert die Funktion das Ergebnis CM_OK zurück. Dieser Aufruf ändert den Zustand der Conversation nicht.

Hinweis

Der Aufruf dient lediglich der besseren Portierbarkeit von CPI-C-Programmen. Selbst wenn er CM_OK zurückliefert, ändert sich *sync_level* nicht. UPIC verwendet intern immer "sync_level=CM_NONE".

Verhalten im Fehlerfall

CM_PROGRAM_STATE_CHECK
Programm ändern.

CM_PROGRAM_PARAMETER_CHECK
Programm ändern.

CM_PRODUCT_SPECIFIC_ERROR
Das Betriebssystem kann nicht genügend Speicherplatz für interne Puffer bereitstellen. Überprüfen Sie Ihr Programm auf zu hohe Speicherplatzanforderung und starten Sie ggf. Ihr System neu.

Funktionsdeklaration: Set_Sync_Level

```
CM_ENTRY Set_Sync_ Level ( unsigned char CM_PTR  conversation_ID,
                           CM_SYNC_LEVEL CM_PTR  sync_level,
                          CM_RETURN_CODE CM_PTR  return_code )
```

Set_TP_Name - TP-Name setzen

Der Aufruf *Set_TP_Name* (CMSTPN) setzt für die Conversation die Characteristics *TP_name* und *TP_name_length*. *TP_name* ist der Transaktionscode eines UTM-Teilprogramms.

Set_TP_Name ändert die Werte, die beim *Initialize_Conversation*-Aufruf aus der Side Information entnommen wurden. Die geänderten Werte gelten nur für die Dauer einer Conversation; die Werte in der Side Information selbst werden nicht verändert.

Der *Set_TP_Name*-Aufruf darf nach *Allocate* nicht mehr ausgeführt werden.

Dieser Aufruf gehört zu den Advanced Functions.

Syntax

CMSTPN (conversation_ID, TP_name, TP_name_length, return_code)

Parameter

→ conversation_ID	Identifikation der Conversation
→ TP_name	UTM-Transaktionscode
→ TP_name_length	Länge von *TP_name*
	Minimum: 1, Maximum: 8
← return_code	Ergebnis des Funktionsaufrufs

Ergebnis (*return_code*)

CM_OK
Aufruf ok

CM_PROGRAM_STATE_CHECK
Der Aufruf ist in diesem Zustand nicht erlaubt.

CM_PROGRAM_PARAMETER_CHECK
Der Wert in *conversation_ID* oder *TP_name* ist ungültig oder der Wert in *TP_name_length* ist kleiner als 1 oder größer als 8.

CM_PRODUCT_SPECIFIC_ERROR
Die UPIC-Instanz konnte nicht gefunden werden.

Falls das Ergebnis nicht CM_OK ist, bleiben *TP_name* und *TP_name_length* unverändert.

Zustandsänderung

Keine Zustandsänderung.

Verhalten im Fehlerfall

CM_PROGRAM_STATE_CHECK
: Programm ändern.

CM_PROGRAM_PARAMETER_CHECK
: Programm ändern.

CM_PRODUCT_SPECIFIC_ERROR
: Das Betriebssystem kann nicht genügend Speicherplatz für interne Puffer bereitstellen. Überprüfen Sie Ihr Programm auf zu hohe Speicherplatzanforderungen und starten Sie ggf. Ihr System neu.

Funktionsdeklaration: Set_TP_Name

```
CM_ENTRY Set_TP_name ( unsigned char CM_PTR  conversation_ID,
                       unsigned char CM_PTR  TP_name,
                            CM_INT32 CM_PTR  TP_name_length,
                      CM_RETURN_CODE CM_PTR  return_code )
```

Specify_Local_Port - TCP/IP-Port der lokalen Anwendung setzen

Der Aufruf *Specify_Local_Port* (CMSLP) setzt die Portnummer der lokalen Anwendung. Der Aufruf überschreibt den Wert, der beim *Enable_UTM_UPIC*-Aufruf zugewiesen wurde. Er darf nach dem *Initialize_Conversation*-Aufruf nicht mehr aufgerufen werden.

Diese Funktion gehört zu den zusätzlichen Funktionen des Trägersystems UPIC; sie ist nicht Bestandteil der CPI-C-Schnittstelle.

X/W *UPIC-Local:*

X/W Der Aufruf *Specify_Local_Port* wird bei der Anbindung über UPIC-L nicht unterstützt.

Syntax

CMSLP (port_number, return_code)

Parameter

→ port_number	legt fest, mit welcher Portnummer sich die lokale Anwendung beim Kommunikationssystem anmeldet Minimum: 0, Maximum:32767
← return_code	Ergebnis des Funktionsaufrufs

Ergebnis (*return_code*)

CM_OK
: Aufruf ok

CM_CALL_NOT_SUPPORTED
: Die Funktion wird nicht unterstützt. Dieser Returncode tritt bei UPIC-L und UPIC-R auf BS2000/OSD auf.

X/W
X/W
X/W
: Bei UPIC-L tritt dieser Returncode immer auf. Er zeigt dem Programm an, dass keine Portnummer vergeben werden kann, da UPIC-L diese Information auf Grund des darunterliegenden Kommunikationssystems nicht benötigt.

B
B
B
B
B
B
: Bei UPIC-R (BS2000) tritt der Returncode nur auf, wenn die UPIC-Bibliothek auf BS2000/OSD zusammen mit CMX eingesetzt wird. Das von UPIC-R verwendete Kommunikationssystem CMX bietet auf BS2000-Systemen keine Möglichkeit, an der Schnittstelle IP-Adressen zur Adressierung der Partner-Anwendung zu übergeben. Wenn die UPIC-Bibliothek die Socketschnittstelle als Kommunikationssystem verwendet, dann tritt der Returncode nie auf.

CM_PROGRAM_STATE_CHECK
: Die Conversation ist nicht im Zustand "Reset".

CM_PRODUCT_SPECIFIC_ERROR
Die UPIC-Instanz konnte nicht gefunden werden.

CM_PROGRAM_PARAMETER_CHECK
Der Wert von *port_number* ist ungültig.

Zustandsänderung

Der Aufruf ändert den Zustand der Conversation nicht.

Hinweis

Die lokale Portnummer ist ein rein formaler Wert, der keinerlei Wirkung hat und dessen Angabe nur aus Gründen der Kompatibilität gepflegt wird. Er sollte weggelassen werden.

Verhalten im Fehlerfall

CM_PROGRAM_PARAMETER_CHECK
Programm ändern.

CM_PROGRAM_STATE_CHECK
Programm ändern.

CM_PRODUCT_SPECIFIC_ERROR
Das Betriebssystem kann nicht genügend Speicherplatz für interne Puffer bereitstellen. Überprüfen Sie Ihr Programm auf zu hohe Speicherplatzanforderung und starten Sie ggf. Ihr System neu.

CM_CALL_NOT_SUPPORTED
Muss kein Fehler sein:

X/W Falls eine Anwendung sowohl für UPIC-L als auch für UPIC-R vorgesehen ist, bedeutet dieser Returncode auf Unix- und Windows-Systemen lediglich, dass die Anwendung mit einer UPIC-L-Bibliothek gebunden ist. Das Programm kann sich diesen Returncode merken und auf weitere Aufrufe zum Setzen von Adressinformationen verzichten.

B Auf BS2000/OSD bedeutet dieser Returncode, dass die Anwendung mit UPIC-R und CMX gebunden ist. Das Programm kann sich diesen Returncode merken und auf den Aufruf *Specify_Local_Port* verzichten.

Funktionsdeklaration: Specify_Local_Port

```
CM_ENTRY Specify_Local_Port ( CM_INT32 CM_PTR  port_number,
                              CM_RETURN_CODE CM_PTR  return_code )
```

Specify_Local_Tsel - T-SEL der lokalen Anwendung setzen

Der Aufruf *Specify_Local_Tsel* (CMSLT) setzt den Wert für die Characteristic *T-SEL* der lokalen Anwendung. Der Aufruf überschreibt den Wert, der beim *Enable_UTM_UPIC*-Aufruf zugewiesen wurde. Er darf nach dem *Initialize_Conversation*-Aufruf nicht mehr aufgerufen werden.

Diese Funktion gehört zu den zusätzlichen Funktionen des Trägersystems UPIC; sie ist nicht Bestandteil der CPI-C-Schnittstelle.



UPIC-Local:



Der Aufruf *Specify_Local_Tsel* wird bei der Anbindung über UPIC-L nicht unterstützt.

Syntax

CMSLT (transport_selector, transport_selector_length, return_code)

Parameter

→ transport_selector
Transport-Selektor der lokalen Anwendung, der dem Kommunikationssystem übergeben wird

→ transport_selector_length
Länge des Transport-Selektors in Byte.
Minimum: 0, Maximum: 8

Wird die Länge des Transport-Selektors mit 0 angegeben, so wird der Name der lokalen Anwendung selbst als Transport-Selektor verwendet.

← return_code
Ergebnis des Funktionsaufrufs

Ergebnis (*return_code*)

CM_OK
Aufruf ok



CM_CALL_NOT_SUPPORTED
Die Funktion wird nicht unterstützt. Dieser Returncode tritt nur bei UPIC-L auf. Er zeigt dem Programm an, dass kein T-SEL vergeben werden kann, da UPIC-L diese Information auf Grund des darunterliegenden Kommunikationssystems nicht benötigt.

CM_PROGRAM_STATE_CHECK
Die Conversation ist nicht im Zustand "Reset".

CM_PRODUCT_SPECIFIC_ERROR
Die UPIC-Instanz konnte nicht gefunden werden.

CM_PROGRAM_PARAMETER_CHECK
Der Wert von *transport_selector_length* ist ungültig.

Zustandsänderung

Der Aufruf ändert den Zustand der Conversation nicht.

Verhalten im Fehlerfall

X/W CM_CALL_NOT_SUPPORTED
X/W Muss kein Fehler sein: Falls eine Anwendung sowohl für UPIC-L als auch für
X/W UPIC-R vorgesehen ist, bedeutet dieser Returncode lediglich, dass die Anwendung
X/W mit einer UPIC-L-Bibliothek gebunden ist. Das Programm kann sich diesen Return-
X/W code merken und auf weitere Aufrufe zum Setzen von Adressinformationen ver-
X/W zichten.

CM_PROGRAM_PARAMETER_CHECK
Programm ändern.

CM_PROGRAM_STATE_CHECK
Programm ändern.

CM_PRODUCT_SPECIFIC_ERROR
Das Betriebssystem kann nicht genügend Speicherplatz für interne Puffer bereitstellen. Überprüfen Sie Ihr Programm auf zu hohe Speicherplatzanforderung und starten Sie ggf. Ihr System neu.

Funktionsdeklaration: Specify_Local_Tsel

```
CM_ENTRY Specify_Local_Tsel (unsigned char CM_PTR  transport_selector,
                                    CM_INT32 CM_PTR  transport_selector_length,
                               CM_RETURN_CODE CM_PTR  return_code )
```

Specify_Local_Tsel_Format - TSEL-Format der lokalen Anwendung setzen

Der Aufruf *Specify_Local_Tsel_Format* (CMSLTF) setzt den Wert für die Characteristic *T-SEL-FORMAT* der lokalen Anwendung. Der Aufruf überschreibt den Wert, der beim *Enable_UTM_UPIC*-Aufruf zugewiesen wurde. Er darf nach dem *Initialize_Conversation*-Aufruf nicht mehr aufgerufen werden.

Diese Funktion gehört zu den zusätzlichen Funktionen des Trägersystems UPIC; sie ist nicht Bestandteil der CPI-C-Schnittstelle.



UPIC-Local:



Der Aufruf *Specify_Local_Tsel_Format* wird bei der Anbindung über UPIC-L nicht unterstützt.

Syntax

CMSLTF (tsel_format, return_code)

Parameter

→ tsel_format
: legt fest, welcher Zeichensatz für den Transport Selektor (TSEL) verwendet wird. Folgende Werte können Sie angeben:

- CM_TRANSDATA _FORMAT
  Der Transport-Selektor wird im TRANSDATA-Format an das Kommunikationssystem übergeben.
- CM_EBCDIC_FORMAT
  Der Transport-Selektor wird im EBCDIC Format an das Kommunikationssystem übergeben.
- CM_ASCII_FORMAT
  Der Transport-Selektor wird im ASCII-Format an das Kommunikationssystem übergeben.

← return_code
: Ergebnis des Funktionsaufrufs

Ergebnis (*return_code*)

CM_OK
  Aufruf ok

X/W CM_CALL_NOT_SUPPORTED
X/W   Die Funktion wird nicht unterstützt. Dieser Returncode tritt nur bei UPIC-L auf. Er
X/W   zeigt dem Programm an, dass kein Format für den Transport-Selektor vergeben
X/W   werden kann, da UPIC-L diese Information auf Grund des darunterliegenden
X/W   Kommunikationssystems nicht benötigt.

CM_PROGRAM_STATE_CHECK
  Die Conversation ist nicht im Zustand "Reset".

CM_PRODUCT_SPECIFIC_ERROR
  Die UPIC-Instanz konnte nicht gefunden werden.

CM_PROGRAM_PARAMETER_CHECK
  Der Wert von *tsel_format* ist ungültig.

Zustandsänderung

Der Aufruf ändert den Zustand der Conversation nicht.

Verhalten im Fehlerfall

CM_PROGRAM_PARAMETER_CHECK
  Programm ändern.

CM_PROGRAM_STATE_CHECK
  Programm ändern.

CM_PRODUCT_SPECIFIC_ERROR
  Das Betriebssystem kann nicht genügend Speicherplatz für interne Puffer bereitstellen. Überprüfen Sie Ihr Programm auf zu hohe Speicherplatzanforderung und starten Sie ggf. Ihr System neu.

X/W CM_CALL_NOT_SUPPORTED
X/W   Muss kein Fehler sein: Falls eine Anwendung sowohl für UPIC-L als auch für
X/W   UPIC-R vorgesehen ist, bedeutet dieser Returncode lediglich, dass die Anwendung
X/W   mit einer UPIC-L-Bibliothek gebunden ist. Das Programm kann sich diesen Return-
X/W   code merken und auf weitere Aufrufe zum Setzen von Adressinformationen ver-
X/W   zichten.

Funktionsdeklaration: Specify_Local_Tsel_Format

```
CM_ENTRY Specify_Local_Tsel_Format ( CM_TSEL_FORMAT CM_PTR  tsel_format,
                                     CM_RETURN_CODE CM_PTR  return_code )
```

Specify_Secondary_Return_Code – Eigenschaften des erweiterten Returncode setzen

Mit dem Aufruf *Specify_Secondary_Return_Code* (CMSSRC) setzt das Programm die Eigenschaft erweiterte Returncodes (secondary return code) der CPI-C-Aufrufe.

Diese Funktion gehört zu den zusätzlichen Funktionen des Trägersystems UPIC; sie ist nicht Bestandteil der CPI-C-Schnittstelle.

Syntax

CMSSRC (return_type, return_code)

Parameter

→ return_type
: Spezifiziert die Eigenschaft erweiterter Returncode der CPI-C-Aufrufe. Folgende Werte können Sie angeben:

  CM_RETURN_TYPE_PRIMARY:
  Die entsprechenden UPIC-Aufrufe geben den erweiterten Returncode zurück.

  CM_RETURN_TYPE_SECONDARY:
  Der erweiterte Returncode kann nur über den CMESRC-Aufruf ausgelesen werden. Die entsprechenden UPIC-Aufrufe geben keinen erweiterten Returncode zurück.

← return_code
: Ergebnis des Funktionsaufrufs.

Ergebnis (*return_code*)

CM_OK
: Aufruf OK

CM_NO_SECONDARY_RETURN_CODE
: Die Eigenschaft secondary return code (erweiterter Returncode) steht nicht zur Verfügung.

CM_PROGRAM_PARAMETER_CHECK
: Der Wert des *return_type* ist ungültig.

CM_PROGRAM_STATE_CHECK
: Das Programm ist im Zustand “Start”.

CM_PRODUCT_SPECIFIC_ERROR
: Die UPIC-Instanz konnte nicht gefunden werden.

Hinweis

Die Funktion kann unmittelbar nach einem *Enable_UTM_UPIC*-Aufruf aufgerufen werden. Sie hat keinerlei Wirkung auf den *Enable_UTM_UPIC*-Aufruf.

Zustandsänderung

Keine Zustandsänderung.

Verhalten im Fehlerfall

CM_PROGRAM_PARAMETER_CHECK
Programm ändern.

CM_PROGRAM_STATE_CHECK
Programm ändern.

CM_PRODUCT_SPECIFIC_ERROR
Das Betriebssystem kann nicht genügend Speicherplatz für interne Puffer bereitstellen. Überprüfen Sie Ihr Programm auf zu hohe Speicherplatzanforderung und starten Sie ggf. Ihr System neu.

CM_NO_SECONDARY_RETURN_CODE
Muss kein Fehler sein. Falls eine UPIC-R Anwendung mit verschiedenen UTM-Partnern kommuniziert, von denen einige erweiterte Returncodes unterstützen können und andere nicht, bedeutet dieser Returncode lediglich, dass die Anwendung mit einer UTM-Anwendung kommunizieren will, die keine erweiterten Returncodes unterstützt. Das Programm kann sich diesen Returncode merken und auf den Aufruf *Extract_Secondary_Return_Code* verzichten.

Funktionsdeklaration: Specify_Secondary_Return_Code

```
CM_ENTRY Specify_Secondary_Return_Code (
                           CM_INT32       CM_PTR  return_type,
                           CM_RETURN_CODE  CM_PTR  return_code )
```

4.10 COBOL-Schnittstelle

Die CPI-C-COBOL Programmschnittstelle entspricht weitgehend der in Abschnitt „CPI-C-Aufrufe bei UPIC“ auf Seite 96 beschriebenen C-Schnittstelle. Aus diesem Grund können Sie diese Beschreibung bei der Erstellung von CPI-C-Programmen in COBOL zu Rate ziehen. In diesem Abschnitt sind die Besonderheiten der COBOL-Schnittstelle bei den Datenstrukturen und den CPI-C-Aufrufen zusammengefasst.

COPY-Element CMCOBOL

Für CPI-C-Anwendungen in COBOL wird das COPY-Element CMCOBOL ausgeliefert, das die Bedingungsvariablen und -namen enthält. CMCOBOL finden Sie nach der Installation des Trägersystems UPIC:

- W unter Windows-Systemen in der Datei *upic-installationsverzeichnis*\`copy-cob`
- X auf Unix-Systemen im Dateiverzeichnis *upic-installationsverzeichnis*`/copy-cobol85`.
- B auf BS2000/OSD-Systemen in der Bibliothek SYSLIB.UTM-CLIENT.061.

CMCOBOL muss mit der COPY-Anweisung in die WORKING-STORAGE-SECTION kopiert werden. Die Namen der Konstanten unterscheiden sich von den C-Namen nur durch den Bindestrich statt Unterstrich, z.B. "CM-SEND-RECEIVED" statt "CM_SEND_RECEIVED".

In CMCOBOL wird für die CPI-C-Schnittstelle wegen der CPI-C-Spezifikation der Name TIME-OUT bzw. TIMEOUT verwendet. Da diese Worte bei Micro Focus reserviert sind, muss dieser Name z.B. mit der Anweisung

```
COPY CMCOBOL REPLACING TIME-OUT BY CPIC-TIMEOUT
```

im Source geändert werden.

CPI-C-Aufrufe in COBOL

Die Funktionsnamen von C und COBOL sind identisch. Für die Parameter der CPI-C-Aufrufe gilt folgendes:

- Die Parameter müssen wie bei COBOL üblich per Adresse ("by reference") übergeben werden.
- Jede Variable der Parameterliste muss mit der Stufennummer 01 beginnen.
- Numerische Daten müssen in dem COMP-Format sein, das auf der jeweiligen Maschine das gleiche Binärformat wie bei C erzeugt.



- Bei COBOL-Aufrufen unter Windows-Systemen sind die für eine dynamische Bibliothek (DLL) vorgegebenen Aufruf-Konventionen zu beachten.

Beispiel

Programmausschnitt mit dem Aufruf Initialize:

```
...
 WORKING-STORAGE-SECTION.
*************************

 COPY CMCOBOL.

...

 PROCEDURE DIVISION.
*************************
...
 CALL "CMINIT" USING CONVERSATION-ID,SYM-DEST-NAME,CM-RETCODE.
```

5 XATMI-Schnittstelle

XATMI ist eine von X/Open standardisierte Programmschnittstelle für einen Communication Resource Manager, der Client-Server-Kommunikation mit Transaktionssicherung ermöglicht.

Grundlage der XATMI-Programmschnittstelle ist die X/Open CAE Specification „Distributed Transaction Processing: The XATMI Specification“ vom November 1995. Die Kenntnis dieser Spezifikation wird im Folgenden vorausgesetzt.

Dieses Kapitel beschreibt die XATMI-Schnittstelle für openUTM-Client-Programme, die das Trägersystem UPIC verwenden.
Informationen zum Trägersystem OpenCPIC finden Sie im Handbuch „openUTM-Client für Trägersystem OpenCPIC“.

Die Beschreibung der XATMI-Schnittstelle ist bis auf wenige Ausnahmen plattformunabhängig, die Ausnahmen sind im Text gekennzeichnet.

Begriffe

In der folgenden Beschreibung werden folgende Begriffe verwendet:

Service
: Eine Service-Funktion, die entsprechend der XATMI-Spezifikation in C oder COBOL programmiert ist.
XATMI unterscheidet zwei Arten von Services: End-Services und Intermediate-Services.

  Ein End-Service ist nur mit seinem Client verbunden und ruft keine anderen Services auf.

  Ein Intermediate-Service ruft einen oder mehrere weitere Services auf.

Client
: Eine Anwendung, die Service-Funktionen aufruft.

Server
: Eine UTM-Anwendung, die Service-Funktionen in C und/oder in COBOL enthält. Die Service-Funktionen können aus mehreren Teilprogrammen bestehen.

Request
: Ein Request ist ein Aufruf an einen Service. Der Aufruf kann entweder von einem Client oder von einem Intermediate-Service aus erfolgen.

Requester Die XATMI-Spezifikation verwendet den Begriff „Requester“ als Bezeichnung für jegliche Anwendung, die einen Service aufruft.
Ein Requester kann sowohl Client wie auch Server sein.

Typisierte Puffer
Puffer für den Austausch von typisierten und strukturierten Daten zwischen Kommunikationspartnern. Durch diese typisierten Puffer ist die Struktur der ausgetauschten Daten dem Trägersystem und der Anwendung implizit bekannt. Sie werden auch im heterogenen Verbund automatisch angepasst (encodiert, decodiert).

5.1 Client-Server-Verbund

Das folgende Bild zeigt einen Client-Server-Anwendungsverbund, bei dem Clients, Server und Requester miteinander kommunizieren. Sie tauschen ihre typisierten Datenstrukturen (**Typisierte Puffer**) nach dem Protokoll der „XATMI U-ASE Definition“ aus.

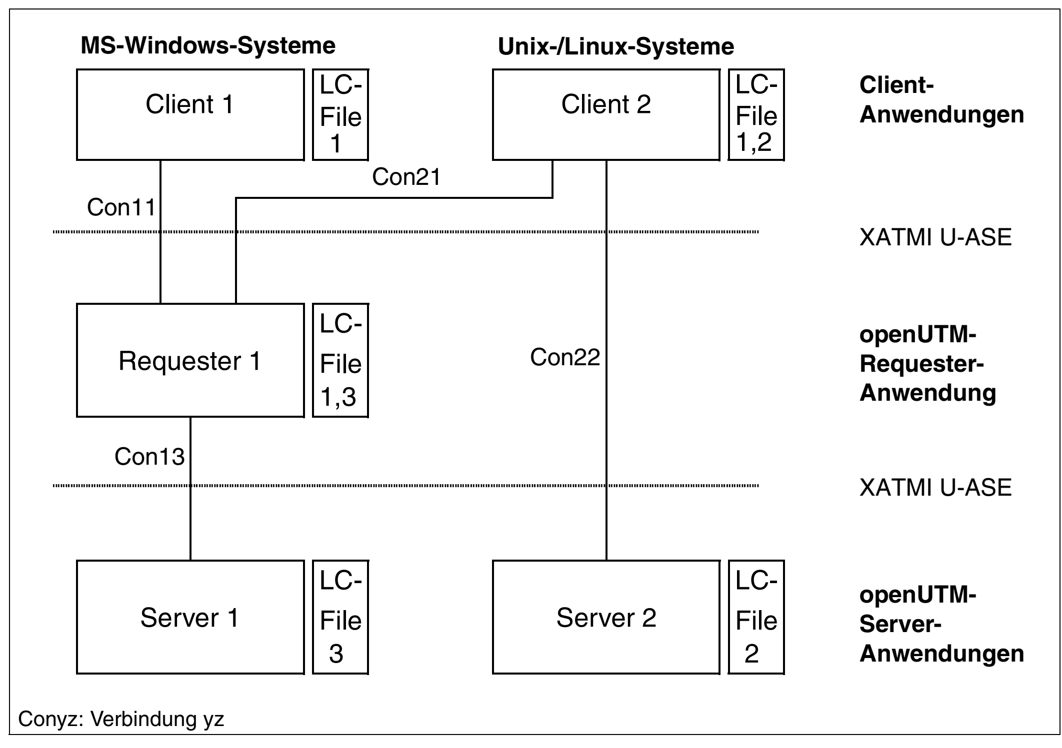


Bild 15: Client-Server-Verbund

In einem beliebigen, heterogenen Anwendungsverbund muss sowohl den Servern wie auch den Clients eine Local Configuration beigestellt sein, die jeweils in der Local Configuration File (LCF) definiert ist. Die Local Configuration beschreibt jeweils die Services und ihre zugehörigen Datenstrukturen, d.h.:

- bei einem Server alle aufrufbaren Services
- bei einem Client die Services aller Server, mit denen der Client in Verbindung steht
- bei einem Requester sowohl alle bereitgestellten als auch alle benutzten Services

Die Local Configurations aller beteiligten Anwendungen müssen aufeinander abgestimmt sein.

Um Client-Server-Verbindungen Con11, Con13,. .. abzuwickeln, stehen mehrere Kommunikationsmodelle zur Verfügung (siehe Abschnitt „Kommunikationsmodelle“ auf Seite 253).

5.1.1 Default-Server

Zur Vereinfachung der Client-Server-Konfiguration bietet Ihnen openUTM-Client die Möglichkeit, mit der Angabe DEST=.DEFAULT in der SVCU-Anweisung der Local Configuration File einen Default-Server zu vereinbaren (siehe Abschnitt „Local Configuration File erzeugen“ auf Seite 270).

Falls bei den Aufrufen `tpcall`, `tpacall` oder `tpconnect` ein Service *svcname2* verwendet wird, der keinen SVCU-Eintrag in der Local Configuration File besitzt, wird automatisch folgender Eintrag verwendet:

```
SVCU svcname2, RSN=svcname2, TAC=scvname2, DEST=.DEFAULT, MODE=RR
```

UPIC erwartet dann in der `upicfile` einen passenden Default-Server-Eintrag, z.B.:

```
LN.DEFAULT localname
SD.DEFAULT servername
```

Zusätzlich besteht die Möglichkeit, einen Service `svcname2@BRANCH9` komplett mit `DEST=BRANCH9` aufzurufen, ohne einen Eintrag in der Local Configuration File anzulegen. In diesem Fall wird folgender Eintrag angenommen:

```
SVCU svcname2, RSN=svcname2, TAC=scvname2, DEST=BRANCH9, MODE=RR
```

Der Partner, in diesem Fall `BRANCH9`, muss dem Trägersystem UPIC bekannt sein. Falls in der Local Configuration File aber ein Eintrag für den Service `svcname2@BRANCH9` vorhanden ist, hat dieser Vorrang gegenüber der Default-Server-Annahme.

5.1.2 Wiederanlauf

Einen Vorgangs-Wiederanlauf gibt es für XATMI zwar nicht (da XATMI keinen Vorgang kennt), aber Sie haben die Möglichkeit einen Recovery-Service zu definieren, der die letzte Ausgabenachricht von openUTM erneut an den Client schickt.

Dieser Recovery-Service wird mit dem Transaktionscode KDCRECVR definiert.

5.2 Kommunikationsmodelle

Für die Client-Server-Kommunikation hat der Programmierer drei Kommunikationsmodelle zur Auswahl:

– Synchrones Request-Response-Modell: Einschritt-Dialog
  Der Client ist nach dem Senden der Service-Anforderung bis zum Eintreffen der Antwort blockiert.
– Asynchrones Request-Response-Modell: Einschritt-Dialog
  Der Client ist nach dem Senden der Service-Anforderung nicht blockiert.
– Conversational Modell: Mehrschritt-Dialog
  Client und Server können beliebig Daten austauschen.

Die für diese Kommunikationsmodelle notwendigen XATMI-Funktionen werden im Folgenden nur skizziert, dabei wird die C-Notation verwendet. Die genaue Beschreibung der XATMI-Funktionen finden Sie in der X/Open-Spezifikation „Distributed Transaction Processing: The XATMI Specification".

Synchrones Request-Response-Modell

Für die Kommunikation wird im Client nur ein einziger *tpcall()*-Aufruf benötigt.

Der *tpcall()*-Aufruf adressiert den Service, schickt genau eine Nachricht an diesen ab und wartet solange, bis ihn die Antwort erreicht, d.h. *tpcall()* wirkt blockierend.

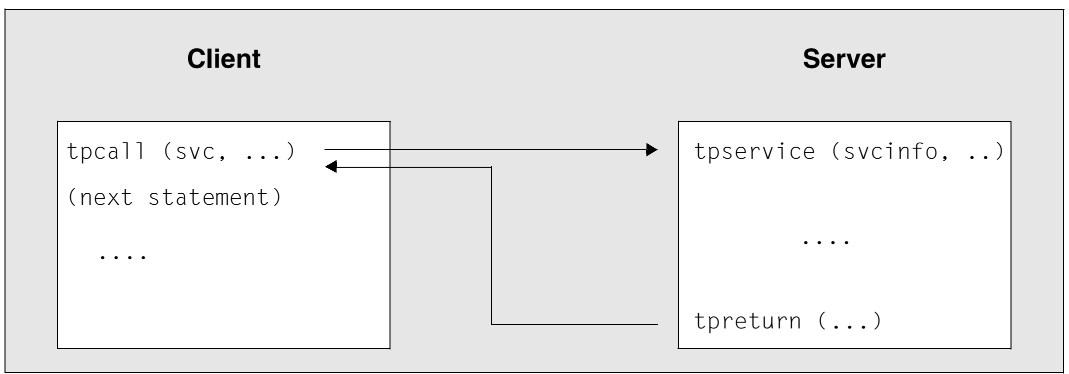


Bild 16: Synchrones Request-Response-Modell

In diesem Bild bezeichnet *svc* den intern verwendeten Namen des Services, *svcinfo* die Service-Info-Struktur mit dem Service-Namen und *tpservice* den Programmnamen der Service-Routine. Die Service-Info-Struktur ist Bestandteil der XATMI-Schnittstelle.

Bei diesem Modell muss auf der UTM-Server-Seite ein Dialog-TAC für den angeforderten Service generiert sein.

Asynchrones Request-Response Modell

Bei diesem Modell wird die Kommunikation in zwei Schritten abgewickelt. Im ersten Schritt wird der Service mit dem Aufruf *tpacall* adressiert und die Nachricht abgeschickt. Zu einem späteren Zeitpunkt wird im zweiten Schritt mit dem Aufruf *tpgetrply* die Antwort abgeholt, siehe folgendes Bild.

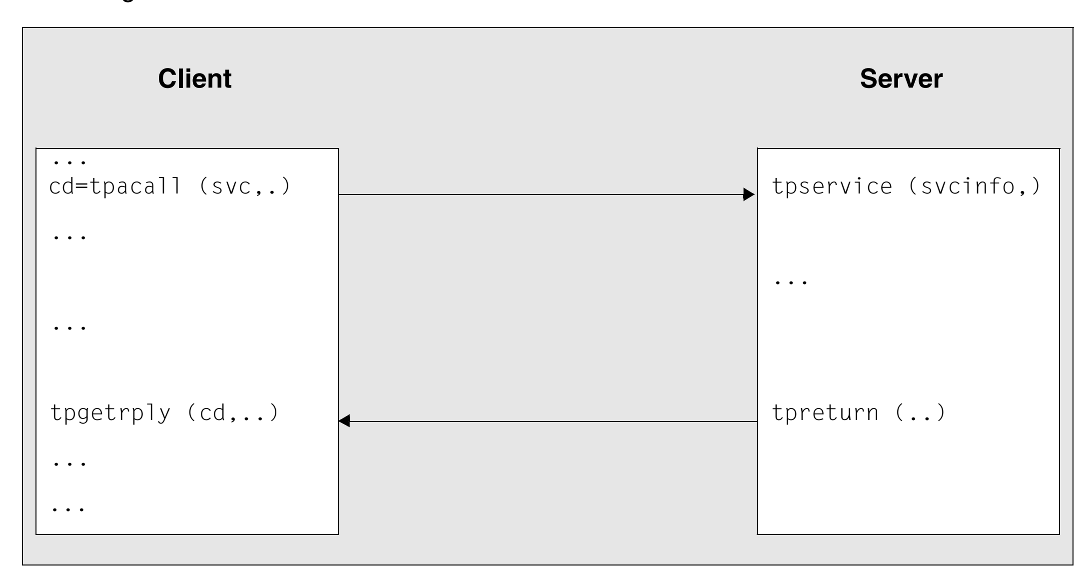


Bild 17: Asynchrones Request-Response-Modell

In dem Bild bezeichnet *svc* den intern verwendeten Namen des Services, *cd* den prozess-lokalen Communication Descriptor, *svcinfo* die Service-Info-Struktur mit dem Service-Namen und *tpservice* den Programmnamen der Service-Routine.

tpacall ist nicht blockierend, d.h. der Client kann in der Zwischenzeit weitere lokale Verarbeitungen durchführen, jedoch keinen weiteren Service parallel aufrufen, da bei Verwendung des Trägersystems UPIC zu einem Zeitpunkt nur ein Auftrag erlaubt ist.
Wenn der Client mehrere Services parallel beauftragen soll, müssen Sie das Trägersystem OpenCPIC verwenden.

tpgetrply hingegegen ist blockierend, d.h. der Client wartet solange, bis die Antwort eingetroffen ist.

Bei diesem Modell muss auf der UTM-Server-Seite für den Service ein Dialog-TAC generiert sein (wie beim synchronen Request-Response).

Conversational Modell

Für verbindungsorientiertes Arbeiten („Conversation“) bietet XATMI das Conversational Modell an.

Dieses Modell kann z.B. verwendet werden, um große Datenmengen in mehreren Teilschritten zu übertragen. Damit können Probleme vermieden werden, die beim synchronen Request-Response Modell (Aufruf *tpcall()*) wegen der Größenbegrenzung der lokalen Datenpuffer auftreten könnten.

Beim Conversational Modell wird die Conversation zu einem Service explizit mit dem Aufruf *tpconnect* aufgebaut. Solange sie besteht, können Client und Server mit *tpsend* und *tprecv* Daten austauschen. Dieser „Dialog“ ist jedoch kein Dialog im Sinne von OSI TP und es kann nur eine Transaktion abgewickelt werden.
Beendet wird die Conversation, wenn der Server mit *tpreturn* das Ende signalisiert; der Client erhält dann beim *tprecv* in der Variablen *tperrno* einen entsprechenden Code. Daher muss das Client-Programm mindestens einen *tprecv*-Aufruf enthalten.

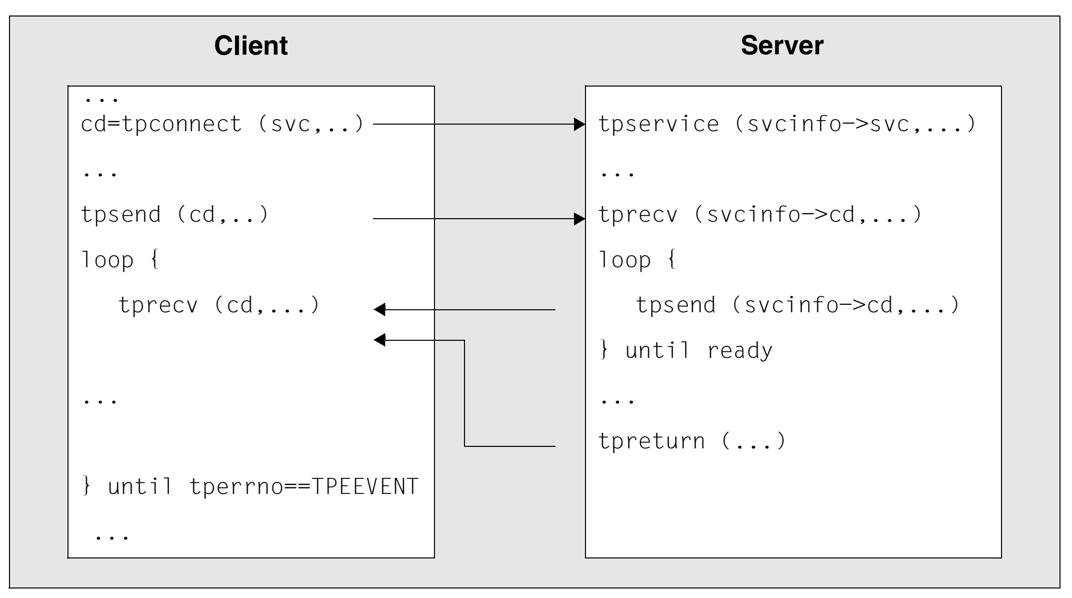


Bild 18: Conversational Modell

In dem Bild bezeichnet *svc* den lokalen Namen des Services, *cd* den prozesslokalen Communication Descriptor, *tpservice* den Programmnamen der Service-Routine und *svcinfo* die Service-Info-Struktur mit dem Service-Namen und dem Communication Descriptor.

Bei dem Modell muss auf der UTM-Server-Seite für den Service ein Dialog-TAC generiert sein.
In Fehlerfällen kann der Client den Abbruch einer Conversation mit dem Aufruf *tpdiscon* erzwingen.

5.3 Typisierte Puffer

XATMI-Anwendungen tauschen Nachrichten mit Hilfe von „typisierten Datenpuffern“ aus. Dadurch werden die über das Netz gehenden Daten korrekt an die Anwendung übergeben, d.h. gemäß der über den Puffernamen identifizierten Datenstruktur mit ihren Datentypen.

Dies hat den Vorteil, dass die Anwendungen keine Maschinenabhängigkeiten berücksichtigen müssen wie z.B. Big Endian/Little Endian Darstellungen, ASCII-/EBCDIC-Konvertierungen oder Ausrichtungen auf Wortgrenzen. Damit können Datentypen wie `int`, `long`, `float` usw. als solche übertragen werden.
Eine eventuell notwendige Kodierung/Dekodierung durch die Anwendungsprogramme entfällt, da dies von XATMI übernommen wird (gemäß den Regeln der XATMI U-ASE Definition).

Ein Datenpuffer-Objekt besteht aus vier Komponenten:

– Typ: Definiert die Klasse des Puffers. Es gibt drei Typen (siehe unten).
– Subtyp: Definiert das Objekt des Typs, d.h. die eigentliche Datenstruktur.
– Längenangabe
– Dateninhalt

Ein solcher Datenpuffer wird während der Laufzeit erzeugt und kann dann über seinen Variablen-Namen (=Subtyp-Name) angesprochen werden. Der Subtyp definiert die Struktur, der Typ legt die Wertemenge der erlaubten elementaren Datentypen fest. In C-Programmen werden solche Puffer dynamisch mit *tpallcoc* erzeugt, man spricht dann von „typisierten Puffern“. In Cobol-Programmen sind diese Puffer statisch festgelegt, man spricht von „typisierten Records“.

Typen

Mit dem Typ eines Datenpuffers wird festgelegt, welche elementaren Datentypen der verwendeten Programmiersprache erlaubt sind. Dadurch wird ein gemeinsames Datenverständnis in einem heterogenen Client-Server-Verbund ermöglicht.

Bei XATMI sind drei Typen definiert:

X_OCTET
: Untypisierter Datenstrom von Bytes („Userbuffer“). Dieser Typ besitzt keine Subtypen. Es wird keine Konvertierung vorgenommen.

X_COMMON
: Alle von C und COBOL gemeinsam verwendbaren Datentypen. Die Konvertierung wird von XATMI vorgenommen.

X_C_TYPE
: Alle elementaren C-Datentypen mit Ausnahme von Zeigern. Die Konvertierung wird von XATMI vorgenommen.

Subtypen

Subtypen haben einen bis zu 16 Zeichen langen Namen, unter dem sie im Anwendungsprogramm angesprochen werden. Jedem Subtyp ist eine Datenstruktur (C-Structure oder COBOL-Record) zugeordnet, die die Syntax des Subtyps bestimmt, siehe Abschnitt „Typisierte Puffer erstellen“ auf Seite 267.
Die Datenstrukturen dürfen nicht geschachtelt werden.

In der Local Configuration wird die Struktur eines Subtyps durch einen Syntaxstring repräsentiert, in dem jeder elementare Datentyp (Basistyp) durch einen Code gekennzeichnet ist, der im Bedarfsfall die Angabe von Feldlängen (<m> und <n>) enthält.

Die folgende Tabelle gibt einen Überblick über die elementaren Datentypen (Basistypen), deren Codes und den Zeichenvorrat der String-Typen:

Code[*]	**Bedeutung**	**ASN.1-Typ**	**X_C_TYPE**	**X_COMMON**
s	short integer	INTEGER	short	S9(4) COMP-5
S<n>	short integer array	SEQUENCE OF INTEGER	short[n]	S9(4) COMP-5 ...
i	integer	INTEGER	integer	--[†]
I<n>	integer array	SEQUENCE OF INTEGER	integer[n]	--
l	long integer	INTEGER	long	S9(9) COMP-5
L<n>	long integer array	SEQUENCE OF INTEGER	long[n]	S9(9) COMP-5 ...
f	float	REAL	float	--
F<n>	float array	SEQUENCE OF REAL	float[n]	--
d	double	REAL	double	--
D<n>	double array	SEQUENCE OF REAL	double[n]	--
c	character	OCTET STRING	char	PIC X
t	character	T.61-String	char	PIC X
C<n>	character array: Alle Werte von 0 bis 255 (dezimal)	OCTET STRING	char[n]	PIC X(n)
C!<n>	character array, durch Null ('\0') terminiert	OCTET STRING	char[n]	--
C<m>:<n>	character matrix[‡]	SEQUENCE OF OCTET STRING	char[m][n]	--
C!<m>:<n>	character matrix, durch Null ('\0') terminiert	SEQUENCE OF OCTET STRING	char[m][n]	--

Code*	Bedeutung	ASN.1-Typ	X_C_TYPE	X_COMMON
T<n>	Die abdruckbaren Zeichen A-Z, a-z und 0-9 plus˜ eine Reihe von Sonderzeichen und Steuerzeichen, siehe Abschnitt „Zeichensätze“ auf Seite 356	T.61-String	t61str[n]	PIC X(n)
T!<n>	character array, durch Null ('\0') terminiert	T.61-String	t61str[n]	--
T<m>:<n>	character matrix	SEQUENCE OF T.61-String	t61str[m][n]	--
T!<m>:<n>	character matrix, durch Null ('\0') terminiert	SEQUENCE OF T.61-String	t61str[m][n]	--

* Dient in der Local Configuration zur Beschreibung der Datenstrukturen

† -- : in X_COMMON nicht vorhanden

‡ eine character matrix ist ein zweidimensionales character array

˜ gemäß CCITT Recommodation T.61 bzw. ISO 6937

Die Zuordnung zwischen Datenstrukturen, Subtypen und gewünschten Services wird in der Local Configuration festgelegt, siehe Abschnitt „Local Configuration File erzeugen“ auf Seite 270.

Zeichensatz-Konvertierung bei X_C_TYPE und X_COMMON

Die Datenpuffer werden im ASCII-Zeichensatz über das Netz geschickt.

Ein Partner kann jedoch eine andere Zeichensatzcodierung als ASCII verwenden, wie z.B. eine BS2000-Anwendung, die EBCDIC verwendet. In diesem Fall konvertiert die XATMI-Bibliothek bei allen eingehenden und abgehenden Daten den ASN.1-Typ *T.61-String*. (Ausnahme: OCTET STRINGs werden nicht konvertiert.)

Daher darf für das Trägersystem keine automatische Konvertierung generiert werden: Für das Trägersystem UPIC **muss** daher in der `upicfile` das entsprechende Kennzeichen generiert werden:



– Unter Unix- und Windows-Systemen ist das **SD**.

B – Unter BS2000/OSD ist das **HD**.

5.4 Programmschnittstelle

Die folgenden Abschnitte geben einen Überblick über die XATMI-Client-Programmschnittstelle für Clients. Eine detaillierte Beschreibung der Programmschnittstelle und der Error- und Returncodes finden Sie in der X/Open-Spezifikation „Distributed Transaction Processing: The XATMI Specification“. Die Kenntnis dieser Spezifikation ist für die Erstellung von XATMI-Programmen unbedingt erforderlich.

Die Programm-Schnittstelle steht sowohl in C als auch in COBOL zur Verfügung.

5.4.1 XATMI-Funktionen für Clients

Die folgenden Tabellen listen alle unter openUTM erlaubten XATMI-Aufrufe auf und beschreiben, in welcher Rolle (C = Client oder S = Server) sie aufgerufen und bei welchem Kommunikationsmodell sie verwendet werden dürfen.

Dazu kommen die beiden UTM-Client-Aufrufe *tpinit* und *tpterm*. Diese beiden Funktionen sind nicht im XATMI-Standard enthalten und dienen zum Anschluss von XATMI an das Trägersystem. Sie sind nachfolgend im Abschnitt „Aufrufe für den Anschluss an das Trägersystem“ auf Seite 260 beschrieben.

Aufrufe für das Request/Response-Modell

C-Aufruf	COBOL-Aufruf	Aufruf im Client/ Server	Beschreibung
tpcall	TPCALL	C	Service-Anforderung im synchronen Request/ Response-Modell
tpacall	TPACALL	C	Service-Anforderung im asynchronen Request/ Response-Modell bzw. Single Request-Modell (Flag TPNOREPLY gesetzt)
tpgetrply	TPGETRPLY	C	Response im asynchronen Request/Response-Modell anfordern
tpcancel	TPCANCEL	C	löscht eine asynchrone Service-Anforderung, bevor die angeforderte Response eingetroffen ist

Tabelle 11: Aufrufe für das Request/Response-Modell

Aufrufe für das Conversational-Modell

C-Aufruf	COBOL-Aufruf	Aufruf im Client/ Server	Beschreibung
tpconnect	TPCONNECT	C	baut eine Verbindung für den Nachrichtenaustausch auf
tpsend	TPSEND	C, S	sendet eine Nachricht
tprecv	TPRECV	C, S	empfängt eine Nachricht
tpdiscon	TPDISCON	C	baut eine Verbindung für den Nachrichtenaustausch ab

Tabelle 12: Aufrufe für das Conversational-Modell

Aufrufe für typisierte Puffer

C-Aufruf	COBOL-Aufruf	Aufruf im Client/ Server	Beschreibung
tpalloc	--	C, S	reserviert Speicherplatz für einen typisierten Puffer
tprealloc	--	C, S	verändert die Größe eines typisierten Puffers
tpfree	--	C, S	gibt einen typisierten Puffer frei
tptypes	--	C, S	ermittelt den Typ eines typisierten Puffers

Tabelle 13: Aufrufe für typisierte Puffer

5.4.2 Aufrufe für den Anschluss an das Trägersystem

Da für openUTM-Clients UPIC und OpenCPIC als Trägersysteme zur Verfügung stehen, muss sich ein XATMI-Anwendungsprogramm beim ausgewählten Trägersystem explizit mit *tpinit* anmelden und mit *tpterm* abmelden, d.h. das XATMI-Programm hat folgenden formalen Aufbau:

1. *tpinit()*
2. XATMI-Aufrufe, z.B. *tpalloc(), tpcall(), tpconnect(), ...tpdiscon()*
3. *tpterm()*

Die beiden Aufrufe *tpinit* und *tpterm* sind im Folgenden beschrieben.

Eine allgemeine Beschreibung des UTM-Benutzerkonzepts finden Sie in Abschnitt „Benutzerkonzept, Security und Wiederanlauf“ auf Seite 77.

tpinit - Client initialisieren

Syntax

```
C:      #include <xatmi.h>

       int tpinit (TPCLTINFO *tpinfo)          (in)


COBOL:  01 TPCLTDEF-REC.
        COPY TPCLTDEF.

        CALL "TPINIT" USING TPCLTDEF-REC.
```

Beschreibung

Die Funktion *tpinit* initialisiert einen Client und identifiziert diesen beim Trägersystem. Sie muss als **erste** XATMI-Funktion in einem Client-Programm aufgerufen werden. Als Parameter ist in C ein Zeiger auf die vordefinierte Struktur *TPCLTINFO* zu übergeben, in COBOL muss der COBOL-Record TPCLTDEF versorgt werden.

C-Struktur TPCLTINFO:

```
#define  MAXTIDENT  9

typedef struct {
    long  flags;                    /* for future use */
    char  usrname[MAXTIDENT];
    char  cltname[MAXTIDENT];
    char  passwd [MAXTIDENT];
} TPCLTINFO;
```

COBOL-Record TPCLTDEF:

```
05 FLAG           PIC S(9) COMP-5.
05 USRNAME        PIC X(9).
05 CLTNAME        PIC X(9).
05 PASSWD         PIC X(9).
```

In *usrname* wird eine Benutzerkennung und in *passwd* ein Kennwort eingetragen. Beide Parameter werden zur Einrichtung einer Conversation verwendet und dienen dazu, auf der UTM-Seite die Zugangsberechtigung nachzuweisen. Mit *cltname* (= local client name) wird der Client beim Trägersystem identifiziert.

cltname ist:



– bei UPIC-L der PTERM-Name oder der lokale Anwendungsname aus der `upicfile`,
– bei UPIC-R der Eintrag in der upicfile oder der TNS-Eintrag (Unix- oder Windows-System, siehe Abschnitt „Konfiguration mit TNS-Einträgen“ auf Seite 296) oder der BCMAP-Eintrag (BS2000/OSD, siehe Abschnitt „Konfiguration mit BCMAP-Einträgen“ auf Seite 296).

Wenn *usrname* und *passwd* mit dem Nullstring initialisiert sind (COBOL: SPACES), dann werden die Security-Funktionen nicht aktiviert, d.h. es findet bei openUTM keine Zugangsprüfung statt. Enthält mindestens einer dieser beiden Parameter einen gültigen Wert, dann wird dieser von openUTM geprüft.
Ist *cltname* mit dem Nullstring bzw. SPACES initialisiert, dann wird der „local client name" mit 8 Leerzeichen vorbelegt.

Wenn *tpinit* in C mit einem NULL-Zeiger aufgerufen wird, dann ist keine Zugangsprüfung aktiviert und der „local client name" ist mit 8 Leerzeichen vorbelegt. Bei COBOL muss dazu die Struktur mit SPACES versorgt werden.

Die Einträge in *usrname*, *passwd* und ggf. in *cltname* müssen den UTM-Namenskonventionen entsprechen, d.h. sie dürfen maximal acht Zeichen lang sein und müssen in C mit dem Stringende-Zeichen ("\0") abgeschlossen sein.

Returnwerte

tpinit liefert im Fehlerfall -1 zurück und setzt die Fehlervariable *tperrno* auf einen der folgenden Werte:

TPEINVAL
: Ein oder mehrere Parameter wurden mit einem ungültigen Wert versorgt.

TPENOENT
: Die Initialisierung konnte nicht durchgeführt werden, z.B. steht nicht genügend Speicherplatz für interne Puffer bereit.

TPEPROTO
: *tpinit* wurde an nicht erlaubter Stelle aufgerufen, z.B. der Client ist bereits initialisiert.

TPESYSTEM
: Es ist ein interner Fehler aufgetreten.

tpterm - Client abmelden

Syntax

```
C:      int tpterm ()

COBOL:  CALL "TPTERM".
```

Beschreibung

Die Funktion *tpterm* meldet den Client, in dem diese Funktion aufgerufen wird, beim Trägersystem ab. Der Client muss zuvor mit *tpinit* initialisiert worden sein.
Nach *tpterm* ist kein XATMI-Aufruf mehr erlaubt, ausgenommen ein erneuter *tpinit*.

Returnwerte

tpterm liefert im Fehlerfall -1 zurück und setzt die Fehlervariable *tperrno* auf einen der folgenden Werte:

TPENOENT
: Der Client konnte sich nicht ordnungsgemäß abmelden. Ursache können z.B. Probleme beim Trägersystem sein.

TPEPROTO
: *tpterm* wurde an einer nicht erlaubten Stelle aufgerufen, d.h. der Client ist noch nicht initialisiert.

TPESYSTEM
: Es ist ein interner Fehler aufgetreten.

5.4.3 Transaktionssteuerung

Beim Aufruf eines XATMI-Services wird vom Client mit dem Aufrufparameter *flag* (in C) bzw. dem Feld TPTRAN-FLAG (in COBOL) gesteuert, ob ein aufgerufener Service in die globale Transaktion eingeschlossen wird.

Für die XATMI-C-Schnittstelle ist die Aufnahme des Service in die globale Transaktion der Standardwert. Soll der Service nicht in die globale Transaktion aufgenommen werden, muss explizit das Flag TPNOTRAN gesetzt werden.
Für die XATMI-COBOL-Schnittstelle gibt es keinen Standardwert, entweder TPTRAN oder TPNOTRAN muss gesetzt werden.

Wird der Service mit dem Flag TPTRAN gestartet, so ist er in die globale Transaktion eingeschlossen.
Beim Aufruf *tpreturn()* wird durch den im Parameter *rval* zurückgelieferten Wert TPSUCCESS bzw. TPFAIL gesteuert, ob die Transaktion erfolgreich beendet oder zurückgesetzt wird.



Wird die XATMI-Schnittstelle mit dem Trägersystem UPIC verwendet, so wird das Flag TPTRAN ignoriert und intern das Flag TPNOTRAN gesetzt. Dieses Verhalten dient zur besseren Portierbarkeit von XATMI-Programmen.

5.4.4 Mischbetrieb

Als Mischbetrieb wird die Kommunikation eines XATMI-Programms mit einem CPI-C-Programm bezeichnet.

Für die Zusammenarbeit mit einem CPI-C-Programm muss das XATMI-Programm die entsprechenden CPI-C-Aufrufe enthalten, der Verbindungsaufbau wird jedoch vom XATMI-Partner durchgeführt. Bei der Kommunikation zu einem Partner muss auf beiden Seiten dieselbe Schnittstelle verwendet werden, d.h. in XATMI-Programmen ist der Aufruf von `Deallocate()` verboten.

5.4.5 Administrationsschnittstelle

In XATMI-Programmen darf nur der KDCS-Aufruf KDCADMI() verwendet werden, andere KDCS-Aufrufe sind nicht erlaubt.

Auf der UTM-Seite müssen bei der KDCDEF-Generierung der entsprechende TAC und evtl. der USER mit Administrationsberechtigung generiert werden.

5.4.6 Include-Dateien und COPY-Elemente

Für die Erstellung von openUTM-Client-Programmen, die die XATMI-Schnittstelle verwenden, werden Include-Dateien für C und COPY-Elemente für COBOL ausgeliefert.

Beim Binden der Client-Programme muss die UTM-Client-Bibliothek eingebunden werden.

C-Module mit XATMI-Aufrufen benötigen folgende Dateien:

1. Die Include-Datei `xatmi.h`.
2. Die Datei(en) mit den Datenstrukturen für alle typisierten Puffer, die im Modul verwendet werden, siehe auch Abschnitt „Typisierte Puffer“ auf Seite 256.

COBOL-Module mit XATMI-Aufrufen benötigen folgende COPY-Elemente und Dateien:

1. Die COPY-Elemente TPSTATUS, TPTYPE, TPSVCDEF und TPCLTDEF.
2. Die Datei(en) mit den Datenstrukturen für alle „typed records“, die im Modul verwendet werden.

W W — i — Auf Windows-Systemen wird die XATMI-Schnittstelle nicht in COBOL unterstützt.

W **Windows-Systeme**

W Unter Windows-Systemen finden Sie die Include-Dateien jeweils im Dateiverzeichnis

W *xatmipfad*`\include`

W *xatmipfad* ist das Verzeichnis, unter dem XATMI installiert ist.
W Das Verzeichnis `C:\Programme\xatmi` ist voreingestellt.

W Es werden keine Copy-Elemente für COBOL ausgeliefert.

X **Unix-Systeme**

X Unter Unix-Systemen finden Sie die Include-Dateien jeweils im Dateiverzeichnis

X *upicpfad*`/xatmi/include`

X und die COPY-Elemente im Dateiverzeichnis

X *upicpfad*`/xatmi/copy-cobol85`

X Die UTM-Client-Bibliothek heißt

X *upicpfad*`/sys/libxtclt.a` bzw. *upicpfad*`/sys/libxtclt.so` (shared objects)

X Dabei steht *upicpfad* für das Dateiverzeichnis, in dem openUTM-Client installiert wurde.

B **BS2000/OSD**

B Unter BS2000/OSD finden Sie die Include-Dateien und die COPY-Elemente als
B Bibliothekselemente vom Typ S in der Bibliothek

B $*userid*.SYSLIB.UTM-CLIENT.061

B Dabei steht $*userid* für die Kennung, unter der openUTM-Client installiert wurde.

5.4.7 Ereignisse und Fehlerbehandlung

Wenn ein Ereignis eingetroffen oder ein Fehler aufgetreten ist, geben XATMI-Funktionen den Returnwert -1 zurück. Zur genaueren Bestimmung von Ereignis oder Fehler muss das Programm die Variable *tperrno* auswerten.

Bei der Conversational-Funktion *tprecv* zeigt *tperrno*=`TPEEVENT` an, dass ein Ereignis eingetroffen ist. Dieses Ereignis kann durch Auswerten des *tprecv*-Parameters *revent* bestimmt werden. Z.B. wird ein erfolgreiches Beenden eines Conversational Services wie folgt angezeigt:

```
Returncode von tprecv =-1
tperrno=TPEEVENT
revent=TPEV_SVCSUCC
```

Bei der Funktion *tpsend* hat der Parameter *revent* keine Bedeutung.

Außerdem kann das Service-Programm beim Ende der Service-Funktion mit *tpreturn* über den Parameter *rcode* einen frei definierten Fehlercode zurückgeben, der im Client über die externe Variable *tpurcode* ausgewertet werden kann, siehe „Distributed Transaction Processing: The XATMI Specification“.

5.4.8 Typisierte Puffer erstellen

Typisierte Puffer werden definiert durch Datenstrukturen in Include-Dateien (bei C) bzw. COPY-Elementen (bei COBOL). Diese Include-Dateien bzw. COPY-Elemente müssen in den beteiligten Programmen eingefügt werden.

Der Datenaustausch zwischen den Programmen erfolgt auf Basis dieser Datenstrukturen, die daher sowohl dem Client als auch dem Server bekannt sein müssen. Dabei sind alle Datentypen erlaubt, die in der Tabelle auf Seite 257 beschrieben sind.

Die Include- bzw. COBOL-COPY-Dateien, in denen die typisierten Puffer beschrieben sind, dienen als Eingabe für das Generierungsprogramm `xatmigen` bzw. `xtgen32`, siehe Abschnitt „Das Tool xatmigen bzw. xtgen32“ auf Seite 275. Für diese Dateien gelten folgende Regeln:

– C- und COBOL-Datenstrukturen müssen in eigenen Dateien stehen. Eine Datei, die sowohl C-Includes als auch COBOL-COPY-Elemente enthält, ist als Eingabe nicht erlaubt.

– Die Dateien dürfen nur aus den Definitionen der Datenstrukturen, Leerzeilen und Kommentaranweisungen bestehen.
Include-Dateien (für C) dürfen auch Makroanweisungen enthalten, d.h. Anweisungen, die mit "#" beginnen.

– Die Datenstrukturen-Definitionen müssen vollständig angegeben werden. Insbesondere müssen COBOL-Datensätze mit der Stufennummer "01" beginnen.

– Die Datenstrukturen dürfen nicht verschachtelt sein.

– Als Feldlängen sind nur absolute Werte und keine Makro-Konstanten erlaubt.

– Es sind nur die Datentypen erlaubt, die in der Tabelle auf Seite 257 beschrieben sind. Insbesondere sind bei C keine Zeiger-Typen zugelassen.

Mit Hilfe des Generierungstools `xatmigen` bzw. `xtgen32` muss der Anwender ggf. die Character-Arrays auf die ASN.1-Stringtypen abbilden, da weder C noch COBOL diese Datentypen kennen, siehe Abschnitt „Das Tool xatmigen bzw. xtgen32“ auf Seite 275.

Für C stehen XATMI-Aufrufe für die Speicherbelegung zur Verfügung (*tpalloc* ...).

Im folgenden finden Sie je ein einfaches Beispiel für C und COBOL.

Beispiel

1. C-Include für typisierten Puffer

```
typedef struct {
        char   name[20];    /* Personenname */
        int    age;         /* Alter        */
        char sex;
        long shoesize;
} t_person;

struct t_city {
        char   name[32];    /* Staedtename */
        char   country;
        long   inhabitants;
        short  churches[20];
        long founded;
}
```

2. COBOL-COPY für Typed Record

```
***** Personal-Record
 01 PERSON-REC.
    05 NAME     PIC      X(20).
    05 AGE      PICTURE  S9(9) COMP-5.
    05 SEX      PIC      X.
    05 SHOESIZE PIC      S9(9) COMP-5.

***** City-Record
 01 CITY-REC.
    05 NAME        PIC   X(32).
    05 COUNTRY     PIC   X.
    05 INHABITANTS PIC   S9(9) COMP-5.
    05 CHURCHES    PIC   S9(4) COMP-5 OCCURS 20 TIMES.
    05 FOUNDED     PIC   S9(9) COMP-5.
```

Weitere Beispiele finden Sie in der X/Open-Spezifikation zu XATMI.

5.4.9 Characteristics von XATMI in UPIC

Dieser Abschnitt beschreibt Besonderheiten, die durch die Implementierung der XATMI-Schnittstelle in openUTM bei Verwendung des Trägersystems UPIC auftreten.

- Es werden alle für Clients relevanten XATMI-Aufrufe unterstützt. Hinzu kommen die beiden Aufrufe *tpinit* und *tpterm*.
- Es ist pro Service nur eine Conversation möglich.
- Es dürfen innerhalb einer Client-Anwendung maximal 100 Pufferinstanzen gleichzeitig verwendet werden. Bei einer Anwendung in C heißt das z.B. maximal 100 *tpalloc*-Aufrufe ohne *tpfree*-Aufruf.
- Die maximale Nachrichtenlänge ist 32000 Byte.

  Die maximale Größe eines typisierten Puffers ist immer kleiner als die maximal mögliche Nachrichtenlänge, da die Nachrichten neben den Nettodaten noch einen „Overhead“ enthalten. Je komplexer ein Puffer ist, desto größer ist der Overhead.

  Als Faustregel gilt: maximale Puffergröße = 2/3 der maximalen Nachrichtenlänge.

  Bei größeren Datenmengen sollte daher immer das Conversational Modell (*tpsend*/*tprecv*) verwendet werden.
- Für die Namenslängen gelten folgende Maximalwerte:

  Servicename 16 Byte

  Puffername 16 Byte

  Nach dem Standard dürfen Servicenamen 32 Byte lang sein, von denen allerdings nur die ersten 16 Byte relevant sind (Konstante XATMI_SERVICE_NAME_LENGTH). Es empfiehlt sich daher, für Servicenamen nicht mehr als 16 Byte zu verwenden.

5.5 Konfigurieren

Für jede XATMI-Anwendung muss der Anwender eine Local Configuration erzeugen. Diese beschreibt die bereitgestellten und genutzten Services mit ihren Zieladressen sowie die verwendeten typisierten Puffer mit ihrer Syntax. Die Information ist in einer Datei hinterlegt, der Local Configuration File (LCF), die von der Anwendung beim Starten einmalig gelesen wird. Eine LCF ist sowohl für die Client- als auch für die Service-Seite notwendig.

5.5.1 Local Configuration File erzeugen

Als Anwender müssen Sie eine Eingabe-Datei erstellen, genannt Local Configuration Definition File. Diese Eingabe-Datei muss aus einzelnen Zeilen aufgebaut werden, für die folgende Syntax gilt:

- Eine Zeile beginnt mit einer SVCU- oder BUFFER-Anweisung und spezifiziert genau einen Service oder einen Subtyp (= typisierten Puffer).
- Zwei Operanden werden durch ein Komma getrennt.
- Eine Anweisungs-Zeile wird durch ein Semikolon (';') abgeschlossen.
- Nimmt eine Anweisung mehr als eine Zeile ein, dann muss jeweils am Zeilenende das Fortsetzungszeichen '\' (Gegenschrägstrich) stehen.
- Eine Kommentarzeile beginnt mit dem '#'-Zeichen.
- Leerzeilen können eingefügt werden, z.B. zur besseren Lesbarkeit.

Aus der Datei, die die Local Configuration Definition enthält, erstellen Sie mit Hilfe des Tools `xatmigen` die eigentliche Local Configuration File („Aufruf von xatmigen“ auf Seite 276).

Im Folgenden werden die SVCU- und die BUFFER-Anweisung beschrieben.

SVCU-Anweisung: Aufrufbaren Service definieren

Eine SVCU-Anweisung beschreibt für den Client die Eigenschaften, die notwendig sind, um einen Service in der Partner-Anwendung aufrufen zu können.

Die SVCU-Anweisung kann bei Verwendung des Trägersystems UPIC entfallen, wenn in der `upicfile` ein Default-Server eingetragen ist, für den gilt
transaction-code = *remote-service-name* = *internal-service-name*.

Default-Server

Zur Vereinfachung der Client-Server-Konfiguration bietet Ihnen openUTM-Client die Möglichkeit, mit der Angabe DEST=.DEFAULT in der SVCU-Anweisung der Local Configuration File einen Default-Server zu vereinbaren.

Falls bei den Aufrufen `tpcall`, `tpacall` oder `tpconnect` ein Service *svcname2* verwendet wird, der keinen SVCU-Eintrag in der Local Configuration File besitzt, wird automatisch folgender Eintrag verwendet:

```
SVCU svcname2, RSN=svcname2, TAC=SCVname2, DEST=.DEFAULT, MODE=RR
```

UPIC erwartet dann in der `upicfile` einen passenden Default-Server-Eintrag, z.B.:

```
LN.DEFAULT localname
SD.DEFAULT servername
```

Zusätzlich besteht die Möglichkeit, einen Service `svcname2@BRANCH9` komplett mit `DEST=BRANCH9` aufzurufen, ohne einen Eintrag in der Local Configuration File anzulegen. In diesem Fall wird folgender Eintrag angenommen:

```
SVCU svcname2, RSN=svcname2, TAC=SCVname2, DEST=BRANCH9, MODE=RR
```

Der Partner, in diesem Fall `BRANCH9`, muss dem Trägersystem UPIC bekannt sein. Falls in der Local Configuration File aber ein Eintrag für den Service `svcname2@BRANCH9` vorhanden ist, hat dieser Vorrang gegenüber der Default-Server-Annahme.

Operator	Operanden	Erläuterung
SVCU	internal-service-name	maximal 16 Byte
	[,RSN=remote-service-name]	Standard: internal-service-name
	[,TAC=transaction-code]	Standard: internal-service-name
	,DEST={ destination-name | .DEFAULT }	Partner-Anwendung
	[,MODE=<u>RR</u>| CV]	RR=Request/Response, Standard CV=Conversation
	[,BUFFERS=(subtype-1,...,subtype-n)]	Standard: kein Subtyp

internal-service-name

maximal 16 Byte langer Name, unter dem ein (ferner) Service im Programm angesprochen wird. Dieser Name muss innerhalb der Anwendung eindeutig sein, d.h. er darf in der LCF nur einmal vorkommen.

Pflichtoperand!

RSN=remote-service-name

maximal 16 Byte langer Name eines Services in der *fernen* Anwendung. Dieser Name wird an die ferne Anwendung übertragen; er darf in der LCF mehrfach vorkommen.

Wird dieser Operand weggelassen, dann setzt `xatmigen` für RSN den Wert *internal-service-name* ein.

TAC=transaction-code

maximal 8 Byte langer Transaktionscode, unter dem der Service in der fernen-Anwendung generiert sein muss.

Wird dieser Operand weggelassen, dann setzt das Tool `xatmigen` für TAC den Wert *internal-service-name* ein und kürzt diesen ggf. auf die ersten 8 Byte.

Mit dem Transaktionscode KDCRECVR kann man einen Recovery-Service definieren, der die letzte Ausgabenachricht von openUTM erneut an den Client schickt.

DEST= destination-name / .DEFAULT

destination-name
Maximal 8 Byte lange Identifikation der Partner-Anwendung.

Dieser Name muss in der `upicfile` als Symbolic Destination Name angegeben werden, siehe Abschnitt „UPIC konfigurieren“ auf Seite 278.

.DEFAULT
Es wird ein Default-Server verwendet.

Pflichtoperand!

MODE=RR / CV
Bestimmt, welches Kommunikationsmodell für den Service verwendet wird:

RR	Request-Response Modell, Standardwert
CV	Conversational Modell

BUFFERS=(subtype-1,...,subtype-n)
Liste von Subtyp-Namen, die an den Service geschickt werden dürfen (der Typ X_OCTET ist immer erlaubt). Jeder Name darf maximal 16 Byte lang sein, wobei alle Zeichen des ASN.1-Typs PrintableString erlaubt sind.

Für jeden hier aufgeführten Subtyp muss eine eigene BUFFER-Anweisung angegeben werden, mit der die Eigenschaften des Subtyps definiert werden (siehe BUFFER-Anweisung).

Der Operand BUFFERS= ist stellungs-sensitiv und muss (falls angegeben) immer der *letzte* Operand der Anweisung sein.

Wird BUFFERS= weggelassen, dann sollten an den Service nur Puffer vom Typ "X_OCTET" gesendet werden (eine Typüberprüfung findet nicht statt).

BUFFER-Anweisung

Eine BUFFER-Anweisung definiert einen typisierten Puffer. Gleichnamige Puffer müssen client- und serverseitig gleich definiert sein.
Mehrfachdefinitionen werden nicht überprüft. Der erste Puffer-Eintrag ist gültig, alle anderen werden ignoriert.

Puffer des Typs "X_OCTET" haben keine besonderen Eigenschaften und benötigen deshalb keine Definition. Typisierte Puffer werden mit folgenden Parametern definiert:

Operator	Operanden	Erläuterung
`BUFFER`	`subtype-name`	maximal 16 Byte
	`[,REC=referenced-record-name]`	Standard: subtype-name
	`[,TYPE=X_COMMON / X_C_TYPE]`	Standard: xatmigen setzt TYPE automatisch

subtype-name
: Maximal 16 Byte langer Name des Puffers, der auch bei der SVCU-Anweisung im Operanden BUFFERS= angegeben werden muss. Der Name muss in der Anwendung eindeutig sein.

REC=referenced-record-name
: Name der Datenstruktur für den Puffer, z.B. ist dies bei C-Strukturen der Name des "typedef" bzw. der "struct-Name".

Wird der Operand weggelassen, dann setzt `xatmigen` REC=*subtype-name* ein.

TYPE=
: Typ des Puffers, näheres siehe Abschnitt „Typisierte Puffer“ auf Seite 256.

Wird der Operand weggelassen, dann setzt `xatmigen` den Typ auf X_C_TYPE oder X_COMMON, je nachdem, welche elementaren Datentypen verwendet wurden.

`xatmigen` erzeugt beim Generierungslauf zusätzlich zwei Operanden mit folgender Bedeutung:

LEN=länge
: Länge des Datenpuffers.

SYNTAX=code
: Syntaxbeschreibung der Datenstruktur in der Code-Darstellung, wie sie in der Tabelle auf Seite 257 aufgeführt ist.

5.5.2 Das Tool xatmigen bzw. xtgen32

Das Tool `xtgen32` bereitet aus einer Datei mit der Local Configuration Definition (LC-Definitionsdatei) und der Datei bzw. den Dateien mit den C- oder COBOL-Datenstrukturen (LC-Description Files) eine Local Configuration File (LCF) auf, siehe folgendes Bild:

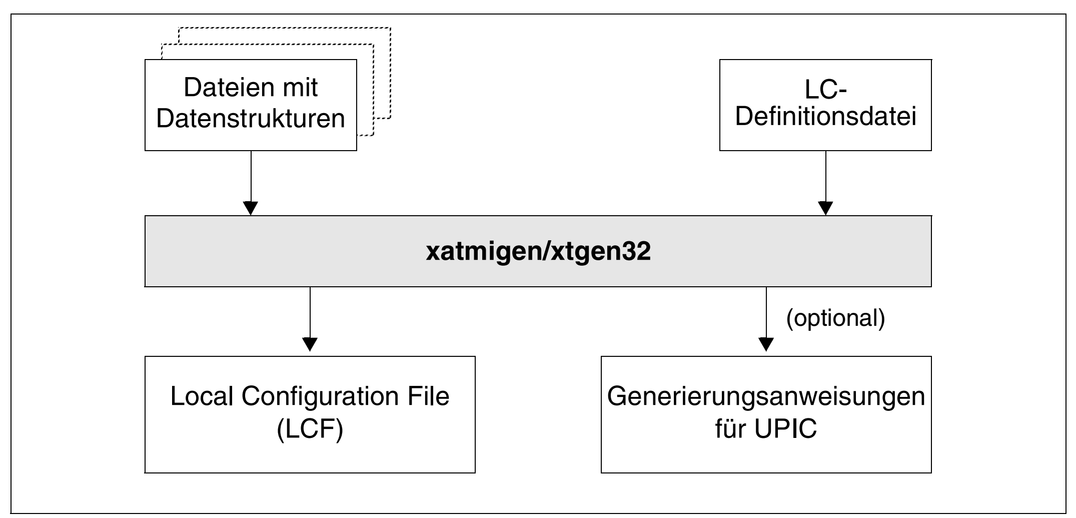


Bild 19: Arbeitsweise von xatmigen/xtgen32

Die Local Configuration File ist gleich aufgebaut wie die LC-Definitionsdatei und unterscheidet sich von dieser nur in der zusätzlichen Beschreibung von Puffertyp, Pufferlänge und Syntaxstring des Puffers. D.h. die BUFFER-Anweisungen werden gegenüber der Definitionsdatei erweitert um die Operanden LEN=, SYNTAX= und ggf. TYPE=.

Falls in der LC-Definitionsdatei der Puffertyp nicht angegeben ist, generiert `xatmigen` den jeweils „kleinsten“ Wertebereich für den Puffertyp, d.h. zuerst den Typ X_COMMON.

Alle Dateinamen müssen explizit angegeben werden. Optional kann eine Datei erstellt werden, die Generierungsanweisungen für UPIC enthält.

W W Unter Windows-Systemen werden Erfolgs- und Fehlermeldungen in das Programmfenster geschrieben.

X X Unter Unix-Systemen werden Erfolgs- und Fehlermeldungen nach `stdout` und `stderr` geschrieben.

B B Unter BS2000/OSD werden Erfolgs- und Fehlermeldungen nach `SYSOUT` und `SYSLST` geschrieben.

Obwohl das Editieren der LCF prinzipiell möglich ist, wird davon dringend abgeraten.

Aufruf von xatmigen

W ● Unter Windows-Systemen wird `xatmigen` aufgerufen mit

W `xtgen32 [.exe] parameter`

W `xtgen32.exe` finden Sie im Dateiverzeichnis *xatmipfad*`\ex.`

W *xatmipfad* ist das Verzeichnis, unter dem XATMI installiert ist. Als Standard ist das
W Verzeichnis `C:\Programme\xatmi32` voreingestellt.

X ● Unter Unix-Systemen wird `xatmigen` aufgerufen mit

X `xatmigen` *parameter*

X `xatmigen` finden Sie im Dateiverzeichnis *upicpfad*/`xatmi`/`ex`.

X Dabei steht *upicpfad* für das Dateiverzeichnis, in dem openUTM-Client installiert wurde.

B ● Unter BS2000/OSD starten Sie `xatmigen` mit folgendem Kommando:

```
/START-PROGRAM $userid.SYSPRG.UTM-CLIENT.061.XATMIGEN
Enter options:
* parameter
```

B *$userid* ist die Kennung, unter der openUTM-Client installiert ist.

B Bei der Eingabe des Kommandos können Sie natürlich statt Großbuchstaben auch
B Kleinbuchstaben verwenden.

Es können folgende Parameter angegeben werden, dabei müssen die Schalter (**-d**, **-l**, **-i**, **-c**) klein geschrieben werden.
Dem Schalter **-d** und, sofern angegeben, den Schaltern **-l** und **-c** muss jeweils der zugehörige Parameter folgen. Die Angabe des Schalters ohne nachfolgenden Parameter ist nicht zulässig.

[upic]
␣**-d**␣lcdf-name
[␣**-l**␣lcf-name]
[␣**-i**]
[␣**-c**␣stringcode]
[␣descript-file-1]. . . [␣descript-file-n]

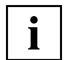
upic Falls angegeben, wird eine Datei `xtupic.def` mit Generierungsanweisungen für die `upicfile` erzeugt. Die Datei wird in das aktuelle Dateiverzeichnis geschrieben.

`upic` muss, sofern angegeben, immer der erste Parameter von `xatmigen` sein. Fehlt die Angabe, dann werden keine Generierungsanweisungen für das Trägersystem UPIC erzeugt.

-d␣lcdf-name
Name der LC-Definitionsdatei, Pflichtangabe.

-l␣lcf-name
Name der zu erzeugenden Local Configuration File. Der Name muss den Konventionen des jeweiligen Betriebssystems entsprechen.

Es wird empfohlen, den Namen maximal 8 Zeichen lang zu wählen und ihn mit der Erweiterung `.lcf` zu versehen.


Eine eventuell vorhandene, gleichnamige LCF wird kommentarlos überschrieben.

Wird der Schalter weggelassen, dann erzeugt `xatmigen` im aktuellen Verzeichnis die Datei `xatmilcf`.

-i
Interaktiver Modus, d.h. bei jedem typisierten Puffer, der ein Character-Array enthält, wird dessen Stringcode erfragt. Die möglichen Angaben für den Stringcode sind bei Schalter **-c** beschrieben.

Der Schalter **-i** hat Vorrang vor einem eventuell ebenfalls vorhandenen Schalter **-c**.

Wenn `xatmigen` im Hintergrund bzw. Batchbetrieb abläuft, dann darf der Schalter **-i** nicht angegeben werden.

-c␣stringcode
Der angegebene Stringtyp gilt für den gesamten `xatmigen`-Lauf, d.h. für alle Character Arrays. Im interaktiven Modus (**-i**) wird Schalter **-c** ignoriert.

Für *stringcode* sind folgende Angaben möglich (siehe Tabelle auf Seite 257):

C Octet-String
C! Octet-String, durch '\0' terminiert
T T.61-String
T! T.61-String, durch '\0' terminiert

Bei fehlender Angabe wird T! eingesetzt.

Auch Einzel-Character werden als T.61-String (*stringcode*= T!) interpretiert.
Die Buchstaben C und T dürfen auch als Kleinbuchstaben angegeben werden.

descript-file-1 . . . descript-file-n
Liste der Dateien, die die Include- bzw. COPY-Elemente mit den Datenstrukturen der typisierten Puffer enthalten.
Fehlt die Angabe, ist nur der Puffertyp X_OCTET zugelassen.

5.5.3 Trägersystem und UTM-Partner konfigurieren

Um eine XATMI-Anwendung funktionsfähig zu machen, müssen Sie

- beim Trägersystem UPIC die UPIC-Konfigurierung (`upicfile`) mit der Local Configuration und der Partner-Generierung abgleichen
- die Initialisierungsparameter, die in *tpinit* angegeben werden, mit der Generierung der UTM-Anwendung abstimmen.

5.5.3.1 UPIC konfigurieren

Für das Trägersystem UPIC muss eine `upicfile` erzeugt werden. Welche Einträge Sie in der `upicfile` machen müssen und wie diese mit der Local Configuration File und der KDCFILE des UTM-Partners korrespondieren, entnehmen Sie Bild 20. Dabei wird TNS-loser Betrieb angenommen. Weitere Informationen finden Sie im Abschnitt „Side Information für stand-alone UTM-Anwendungen“ auf Seite 298.

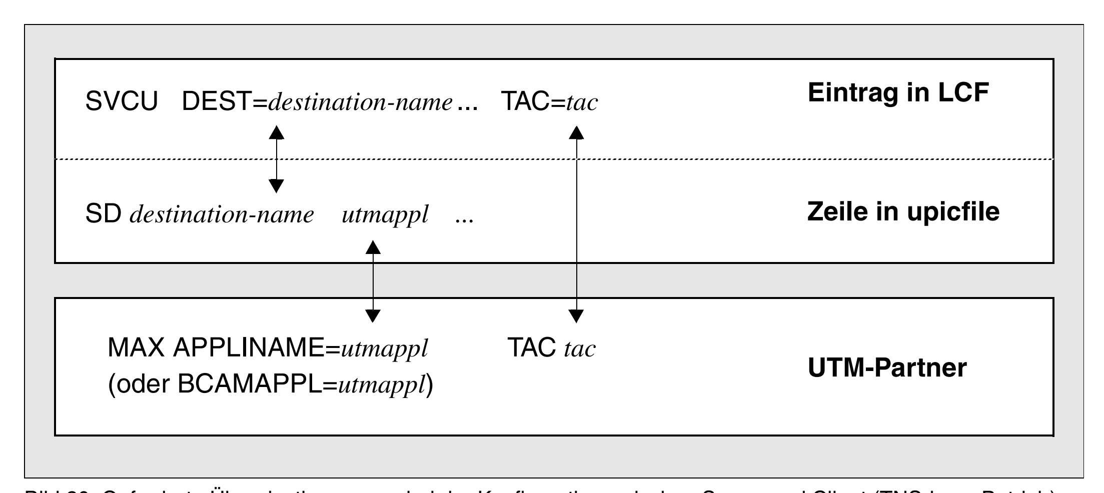


Bild 20: Geforderte Übereinstimmungen bei der Konfiguration zwischen Server und Client (TNS-loser Betrieb)



Ein Eintrag muss mit **SD** (Unix- und Windows-Systeme) beginnen, da XATMI für die Code Konvertierung zwischen ASCII und EBCDIC sorgt. Der *destination-name* muss in LCF und `upicfile` übereinstimmen.

utmappl ist der Name der UTM-Anwendung, wie er in den KDCDEF-Anweisungen MAX APPLINAME oder BCAMAPPL= generiert ist. Die Adress-Informationen wie z.B. IP-Adresse und Portnummer müssen in der upicfile angegeben werden..

Der Transaktionscode *tac* in der SVCU-Anweisung muss mit einer TAC-Anweisung in der UTM-Generierung definiert sein.

Wenn Sie bei `xatmigen` den Parameter *upic* angeben, wird eine `upicfile` erzeugt, bei der die einzelnen Zeilen nur noch um den Parameter *partner* ergänzt werden müssen (per Editor). Wenn Sie den Parameter *upic* nicht angeben, müssen Sie die komplette `upicfile` selbst erstellen.

5.5.3.2 Initialisierungsparameter und UTM-Generierung

Ein XATMI-Client wird mit der Funktion *tpinit* initialisiert. In der Struktur `TPCLTINIT` werden Parameter für Benutzerkennung und Kennwort sowie für den lokalen Anwendungsnamen übergeben. Diese Parameter müssen wie folgt mit der UTM-Generierung abgestimmt sein.

Benutzerkennung und Kennwort

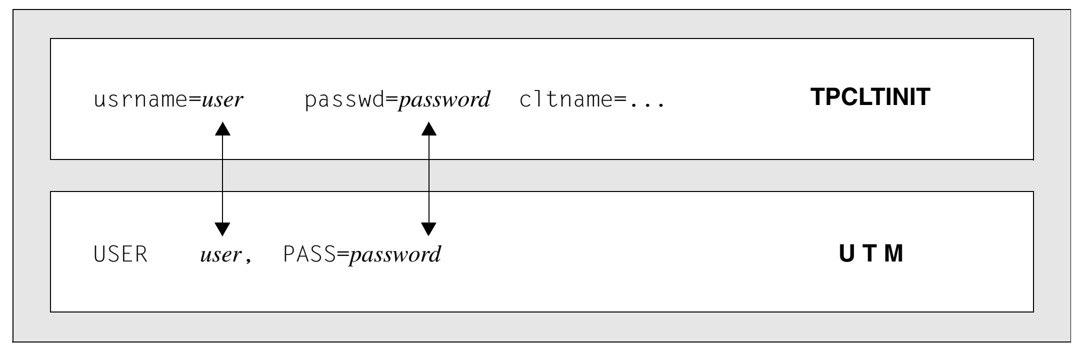


Bild 21: Abstimmung der Generierungsparameter

Falls beim Trägersystem UPIC mit Zugangsprüfung bei openUTM gearbeitet wird, müssen *user* und ggf. *password* sowohl beim Aufruf *tpinit* als auch in einer USER-Anweisung der UTM-Generierung angegeben werden.

Lokaler Anwendungsname

Das folgende Bild zeigt die Initialisierung für den Fall, dass in der `upicfile` ein lokaler Anwendungsname definiert ist (TNS-loser Betrieb über RFC1006).

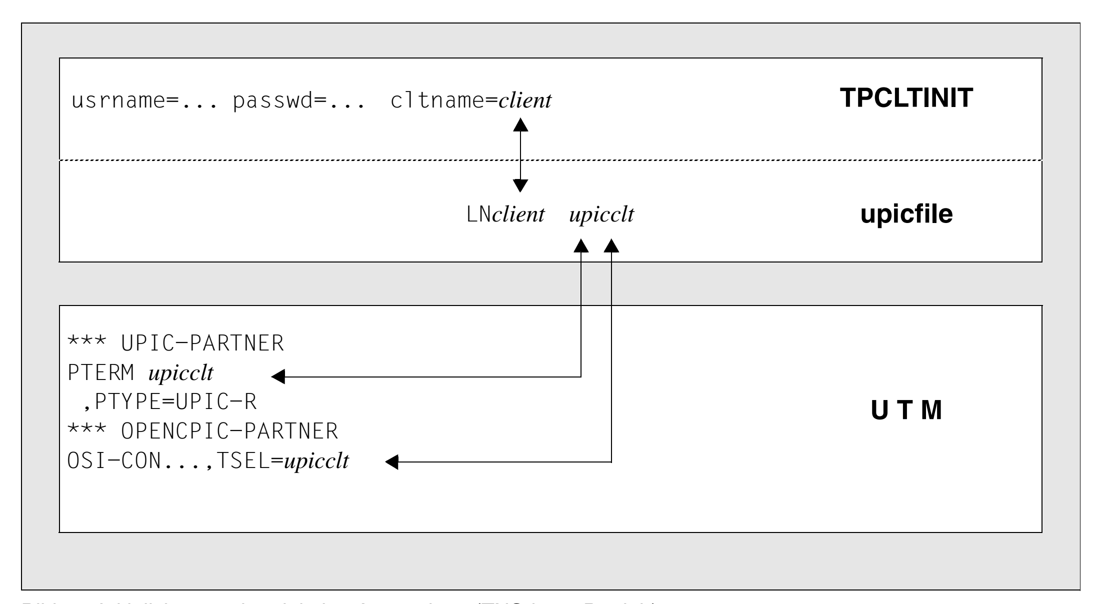


Bild 22: Initialisierung einer lokalen Anwendung (TNS-loser Betrieb)

Ist ein lokaler Anwendungsname in der `upicfile` generiert, dann kann dieser Name beim *tpinit* angegeben werden, in diesem Beispiel *client*. Der zugehörige Anwendungsname muss dann mit dem in der PTERM-Anweisung bzw. bei OSI-CON TSEL= angegebenen Namen übereinstimmen.

Ist kein lokaler Anwendungsname in der `upicfile` generiert, dann muss der Name angegeben werden, der auf UTM-Seite in der PTERM-Anweisung bzw. bei OSI-CON TSEL= definiert ist (in diesem Beispiel *upicclt*).

Beispiel

Das folgende Beispiel umfasst als Auszug alle relevanten Teile von Local Configuration, UPIC-Konfigurierung, Initialisierung und KDCDEF-Generierung.

1. Client

   Local Configuration:

   ```
   SVCU ...
        ,RSN=SERVICE1
        ,TAC=TAC1
        ,DEST=SATURNUS

         ...
   ```

   upicfile:

   ```
   SDSATURNUS utmserv1
   ```

   Initialisierung

   ```
   TPCLTINIT tpinfo;

   strcpy (tpinfo.cltname, "CLIENT1");
   strcpy (tpinfo.usrname, "UPICUSER");
   strcpy (tpinfo.passwd, "SECRET");

   tpinit (tpinfo);
   ```

2. Server

   Local Configuration

   ```
   SVCP SERVICE1  ...    (auch REQP möglich)
        ,TAC=TAC1
   ```

   KDCDEF-Anweisungen

   ```
   MAX APPLINAME=UTMSERV1
   ```

   oder

   ```
   BCAMAPPL UTMSERV1 (Im BS2000 zusätzlich mit Parameter TPROT=ISO)

   LTERM UPICTERM

   PTERM TNSCLIENT, PTYPE=UPIC-R, PRONAM=DxxxSyyy (bei UPIC-Remote-Kopplung)
   PTERM CLIENT1, PTYPE=UPIC-L                    (bei UPIC-Local-Kopplung)

   TAC TAC1, PROGRAM=..., API=(XOPEN,XATMI)

   USER UPICUSER,PASS=SECRET
   ```

5.6 Einsatz von XATMI-Anwendungen

5.6.1 Binden und Starten eines XATMI-Programms

5.6.1.1 Binden eines XATMI-Programms unter Windows-Systemen

Es wird empfohlen, das XATMI-Programm mit der Option __STDC__ (ANSI) zu übersetzen. Zu einer XATMI-Client-Anwendung müssen Sie folgende Bibliotheken mit dazubinden:

1. Alle Clientmodule mit Hauptprogramm
2. Die XATMI-Client-Bibliothek *xatmipfad*\SYS\XTCLT32.LIB

   xatmipfad ist der Pfadname, unter dem XATMI installiert wurde.

Die UPIC-DLLs und die PCMX-DLL müssen verfügbar sein.

3. Wenn Sie XATMI mit UPIC-L auf Windows betreiben wollen, müssen Sie die Bibliothek `libxtclt.lib` zu Ihrem Anwendungsprogramm hinzubinden.

5.6.1.2 Binden eines XATMI-Programms auf Unix-Systemen

Beim Binden einer XATMI-Client-Anwendung müssen folgende Bibliotheken mit dazugebunden werden.

1. Alle Clientmodule mit Hauptprogramm
2. XATMI-Client-Bibliothek und UPIC-Bibliothek (siehe unten)
3. `-lm` (Abkürzung für die "mathlib" auf Unix-Systemen)

Je nachdem, ob UPIC-L oder UPIC-R verwendet wird, sind die folgenden statischen oder dynamischen XATMI- und Trägersystem-Bibliotheken zu binden:

– Trägersystem UPIC-Lokal:

   a) *utmpfad*`/upicl/xatmi/sys/libxtclt.a` oder `.../libxtclt.so`

   b) *utmpfad*`/upicl/sys/libupicipc.a` oder `.../libupicicp.so`

   utmpfad ist der Pfadname, unter dem openUTM installiert wurde.

X – Trägersystem UPIC-Remote:

X a) *upicpfad*/xatmi/sys/libxtclt.a oder .../libxtclt.so

b) CMX: *upicpfad*/sys/libupiccmx.a oder .../libupiccmx.so
Socket: *upicpfad/sys/*libupicsoc.a oder .../libupicsoc.so

X c) CMX-Bibliothek

X Dabei steht *upicpfad* für das Dateiverzeichnis, in dem openUTM-Client installiert wurde.

5.6.1.3 Binden eines XATMI-Programms unter BS2000/OSD

B Beim Binden einer XATMI-Client-Anwendung müssen folgende Bibliotheken mit dazu-
B gebunden werden:

B 1. Alle Clientmodule mit Hauptprogramm

B 2. Die XATMI-Client- und UPIC-Bibliothek *$userid*.SYSLIB.UTM-CLIENT.061

B *$userid* ist die Kennung, unter der UPIC-R installiert wurde.

B In der Bibliothek *$userid*.SYSLIB.UTM-CLIENT.061 finden Sie das Beispiel
B BIND-TPCALL zum Binden eines XATMI-Programms

5.6.1.4 Starten

Ein XATMI-Clientprogramm wird als ausführbares Programm gestartet.

5.6.2 Umgebungsvariablen auf Windows- und Unix-Systemen setzen

X/W Für XATMI-Anwendungen werden von openUTM-Client eine Reihe von Umgebungsvaria-
X/W blen ausgewertet. Die Umgebungsvariablen müssen vor dem Start der Anwendung gesetzt
X/W werden.

X/W Zur Diagnose bei laufender Anwendung können Traces eingeschaltet werden.

X/W **Umgebungsvariablen**

X/W Für eine XATMI-Anwendung werden folgende Umgebungsvariablen ausgewertet:

X/W XTPATH Pfadname für die Trace-Dateien.

X/W Ist diese Variable nicht gesetzt, dann werden die Trace-Dateien in das
X/W aktuelle Verzeichnis geschrieben (= Verzeichnis, unter dem die XATMI-
X/W Anwendung gestartet wurde).

X/W X/W X/W	XTLCF	Dateiname der verwendeten Local Configuration File (LCF). Der Dateiname der Local Configuration File muss den Konventionen des Betriebssystems entsprechen.
X/W X/W		Falls XTLCF nicht gesetzt ist, wird im aktuellen Dateiverzeichnis unter dem Namen `xatmilcf` gesucht.
X/W X/W X/W X/W	XTPALCF	Definiert den Suchpfad für zusätzliche Beschreibungen von typisierten Puffern. Die Pufferbeschreibungen werden aus Local Configuration Files mit dem Namen `xatmilcf` bzw. dem in XTLCF festgelegten Namen gelesen.
X/W X/W		Alle wichtigen XATMI-Generierungen (z.B. SVCU ...) werden auch weiterhin in der über XTLCF festgelegten Local Configuration File gesucht.
X/W X/W X/W X/W		Alle in XTPALCF angegebenen Dateiverzeichnisse werden nach Local Configuration Files durchsucht und die Beschreibungen der typisierten Puffer werden intern gesammelt (bei Namensgleichheit wirkt nur die erste Pufferbeschreibung).
X/W X/W		Der Suchpfad wird genauso aufgebaut wie in der auf Unix-Systemen üblichen Variablen PATH (*verzeichnis1*:*verzeichnis2*: ...).
X/W X/W		Falls der angegebene Pfad die Länge von 1024 Zeichen überschreitet, wird er abgeschnitten. Es sind maximal 128 LCF-Einträge möglich.
X/W	XTSVRTR	Tracemodus für die XATMI-Anwendung. Mögliche Angaben:
X/W	E	(Error): Aktiviert den Fehlertrace.
X/W	I	(Interface): Aktiviert den Schnittstellentrace für die XATMI-Aufrufe.
X/W	F	(Full): Aktiviert den vollen XATMI-Trace sowie den UPIC-Trace.

W **Umgebungsvariabeln auf WIndows-Systemen setzen**

W W W W Unter Windows-Systemen setzen Sie die Umgebungsvariablen über *Start/Einstellungen/ Systemsteuerung* und wählen anschließend *System/Erweitert/Umgebungsvariablen*. Erzeugen bzw. erweitern Sie dort die Umgebungsvariablen. Diese Einstellungen bleiben unter Windows-Systemen bis zur nächsten Änderung gültig.

X **Umgebungsvariabeln auf Unix-Systemen setzen**

X Umgebungsvariablen werden auf Unix-Systemen mit folgendem Kommando gesetzt:

X `SET` *variablenname* = *wert*

X X Die Umgebungsvariablen gelten jeweils für eine Shell; für eine Anwendung in einer anderen Shell können andere Werte gelten.

5.6.3 Jobvariablen setzen unter BS2000/OSD

Für eine XATMI-Anwendung können Jobvariablen gesetzt werden, die über folgende Link-Namen (Kettungsnamen) mit der Anwendung verbunden werden:

XTPATH	Link auf Jobvariable mit dem Prefix für die Namen der Trace-Dateien. Ist dieser Link-Name keiner Jobvariablen zugeordnet, dann werden die Trace-Dateinamen ohne Prefix gebildet.
XTLCF	Link auf Jobvariable mit dem Dateinamen für die Local Configuration File(LCF). Der Dateiname der Local Configuration File muss den Konventionen des Betriebssystems entsprechen.Die Datei wird unter der aktuellen Benutzerkennung gesucht. Ist XTLCF keiner Jobvariablen zugeordnet, dann wird in der aktuellen Benutzerkennung unter dem Namen XATMILCF gesucht.
XTPALCF	Link auf Jobvariable mit dem Suchpfad für zusätzliche Beschreibungen von typisierten Puffern. Die Pufferbeschreibungen werden aus Local Configuration Files mit dem Namen XATMILCF bzw. dem über XTLCF festgelegten Namen gelesen. Alle wichtigen XATMI-Generierungen (z.B. SVCU ...) werden auch weiterhin in dem über XTLCF festgelegten Local Configuration File gesucht. Unter allen im Suchpfad angegebenen Kennungen wird nach Local Configuration Files gesucht und die Beschreibungen der typisierten Puffer aus diesen Dateien werden intern gesammelt (bei Namensgleichheit wirkt nur die erste Puffer-Beschreibung). Der Suchpfad wird in der Form *kennung1:kennung2:...* angegeben.
XTSVRTR	Link auf Jobvariable mit dem Tracemodus für die XATMI-Server-Anwendung. Mögliche Angaben:
E	(Error): Aktiviert den Fehlertrace
I	(Interface): Aktiviert den Schnittstellentrace für die XATMI-Aufrufe
F	(Full): Aktiviert den vollen XATMI-Trace sowie den UPIC-Trace

Wenn das Software-Produkt JV als Subsystem geladen ist, können die Jobvariablen z.B. unter BS2000/OSD wie folgt gesetzt werden:

1. Jobvariable erzeugen:

   CREATE-JV JV-NAME=FULLTR

2. Wert an die Jobvariable übergeben:

   MODIFY-JV JV[-CONTENTS]=FULLTR, SET-VALUE=‘F‘

B 3. Task-spezifischen Jobvariablen-Link setzen:

B SET-JV-LINK LINK-NAME=XTSVRTR, JV-NAME=FULLTR

B 4. Task-spezifischen Jobvariablen-Link anzeigen:

B SHOW-JV-LINK JV[-NAME]=FULLTR

B 5. Task-spezifischen Jobvariablen-Link löschen:

B REMOVE-JV-LINK LINK-NAME=XTSVRTR

B Unter BS2000/OSD sind die Jobvariablen Auftrags-spezifisch. Einer zweiten Anwendung
B unter der gleichen Kennung können andere Jobvariablen zugewiesen werden.

5.6.4 Trace

Jeder Client-Prozess schreibt den Trace in eine eigene Datei, die in zwei Generationen (alt und neu) existieren kann.

Die maximale Größe einer Tracedatei beträgt 128 KB. Sobald diese Größe erreicht wird, wird auf eine zweite Datei umgeschaltet. Hat auch diese das Limit erreicht, wird wieder in die erste Datei geschrieben. Eine Tracedatei besitzt bei einem Client folgenden Namen:

X/W – Unix- und Windows-Systeme:

X/W XTC*pid.n*

X/W XTC Kennzeichnet einen XATMI-Client-Trace

X/W pid Prozess-ID des Client-Prozesses, 4- oder 5-stellig

X/W n Nummer der Generation: 1 oder 2
X/W Den jüngeren Trace erkennen Sie anhand der Zeitstempel.

B – BS2000/OSD:

B `[prefix.]`XTC*tsn.n*

B prefix Der über den Link-Namen XTPATH in der entsprechenden Jobvariable
B vergebene Namensteil (ohne abschließenden Punkt).

B XTC Kennzeichnet einen XATMI-Client-Trace

B tsn ID der Client-Task, 4-stellig

B n Nummer der Generation: 1 oder 2
B Den jüngeren Trace erkennen Sie anhand der Zeitstempel.

Beispiel: XTC00341.1: Client-Tracedatei Nummer 1
XTC00341.2: Client-Tracedatei Nummer 2

5.7 Meldungen des Tools xatmigen

Die Meldungen von XATMIGEN haben die Form `XGnn meldungstext...` und werden unter Windows-Systemen in das Programmfenster, auf Unix-Systemen nach `stderr` und auf BS2000/OSD nach SYSLST ausgegeben.

X/W Auf Windows- und Unix-Systemen können Sie mit der Umgebungsvariablen LANG steuern, ob Sie deutsche oder englische Meldungen erhalten.

B Unter BS2000/OSD können Sie einer Task-spezifischen Jobvariablen mit dem Link-Namen LANG das Sprachkennzeichen 'D' oder 'E' zuweisen und damit steuern, ob Sie deutsche oder englische Meldungen erhalten.

XG01 `Generierung des Local Configuration Files: &LCF / &DEF / &CODE`

Bedeutung
Startmeldung des Tools.

&LCF	Name des erzeugten Local Configuration Files
&DEF	Name des erzeugten Generierungsfragments
&CODE	Stringcode für Character Arrays

XG02 `Generierung erfolgreich beendet`

Bedeutung
Die LCF wurde erzeugt, die Generierung wurde erfolgreich beendet.

XG03 `Generierung erfolgreich mit Warnungen beendet`

Bedeutung
Die LCF wurde erzeugt. Es wird jedoch eine Warnung ausgegeben, da z.B. nicht benötigte Dateien angegeben wurden. Diese Warnung hat allerdings auf die Generierung keinen Einfluss.

XG04 `Generierung wegen Fehlers beendet.`
`Keine Datei erzeugt.`

Bedeutung
Die LCF wurde nicht erzeugt, die Generierung konnte nicht durchgeführt werden. Die Ursache ist vorhergehenden Meldungen zu entnehmen.

XG05 `&FTYPE Datei'&FNAME'`

Bedeutung
Diese Meldung gibt die gerade bearbeitete Datei an in folgender Form:

&FTYPE:	„Description“-File enthält Datenstrukturen
	„Definition“-File enthält den LCF-Input
	„LC“-File enthält die Local Configuration
&FNAME:	Filename

XG10 `Aufruf: &PARAM`

Bedeutung
Syntaxfehler beim Aufruf von XATMIGEN:
&PARAM: Mögliche Aufrufparameter und Schalter

XG11 `[Error] &FTYPE File 'FNAME' kann nicht erzeugt werden.`
`&REASON`

Bedeutung
Die Datei &FNAME des Typs &FTYPE kann nicht erzeugt werden.
&REASON enthält eine nähere Begründung.
&FTYPE: GEN = Generation Fragment File (=Generierungs-Anweisungen)
LC = Local Configuration File

XG12 `[Warning] Datei nicht gefunden.`

Bedeutung
Die Definition File oder eine Description File wurde nicht gefunden; möglicherweise existiert die Datei nicht.

XG13 `[Warning] Zu viele &OBJECTS, Maximum: &MAXNUM`

Bedeutung
Meldung über zu viele gefundene Objekte
&OBJECTS: Subtypen
&MAXNUM: Maximale Anzahl

XG14 `[Error] Zeile &LINE: Syntaxfehler, &HELPTEXT`

Bedeutung
Syntaxfehler in Zeile &LINE in der LC-Definition-Datei
&HELPTEXT: Hilfetext

XG15 `[Error] Zeile &LINE: Keine Record-Definition gefunden für Puffer &BUFF`

Bedeutung
Für den Puffer &BUFF in Zeile &LINE konnte keine zugehörige Record-Definition gefunden werden.

XG16 `[Error] Zeile &LINE: Basistyp-Fehler in Puffer &BUFF`

Bedeutung
Die Syntaxbeschreibung des Puffers &BUFF in Zeile &LINE der LCF enthält einen falschen Basistype (int, short usw.)

XG17 `[Error] &FTYPE File '&FNAME'kann nicht geöffnet werden. &REASON`

Bedeutung
Die Datei &FNAME des Typs &FTYPE kann nicht geöffnet werden.
&REASON enthält eine nähere Begründung.
&FTYPE: DEF (= LC-Definition File)

XG18 `[Error] &REASON`

Bedeutung
Allgemeiner Fehler.
&REASON enthält eine nähere Begründung.

XG19 `[Message] Neuen Puffer erzeugt: '&BUFF'`

Bedeutung
&BUFF: Erzeugter Puffer

XG20 `[Message] Servicename '&SVC' auf 16 Zeichen gekuerzt!`

Bedeutung
&SVC : Servicename.

XG21 `[Message] Zeile &LINE: unbekannte Anweisungszeile '&HELPTEXT'`

Bedeutung
Meldung für die Zeile &LINE in der LC-Definition-Datei
&HELPTEXT: Hilfetext (ein Teil der LC-Zeile)

XG22 `[Message] Zeile &LINE: Standardwert gesetzt MODE='&TEXT'`

Bedeutung
Meldung für die Zeile &LINE in der LC-Definition-Datei
&TEXT: gesetzter Default Servicemode

6 Konfigurieren

Ein Client mit Trägersystem UPIC verwendet als Server immer UTM-Anwendungen in Windows-Systemen, Unix-Systemen oder BS2000/OSD. Daher muss die Konfiguration des Trägersystems UPIC mit der Generierung der UTM-Partner-Anwendung(en) abgestimmt werden.

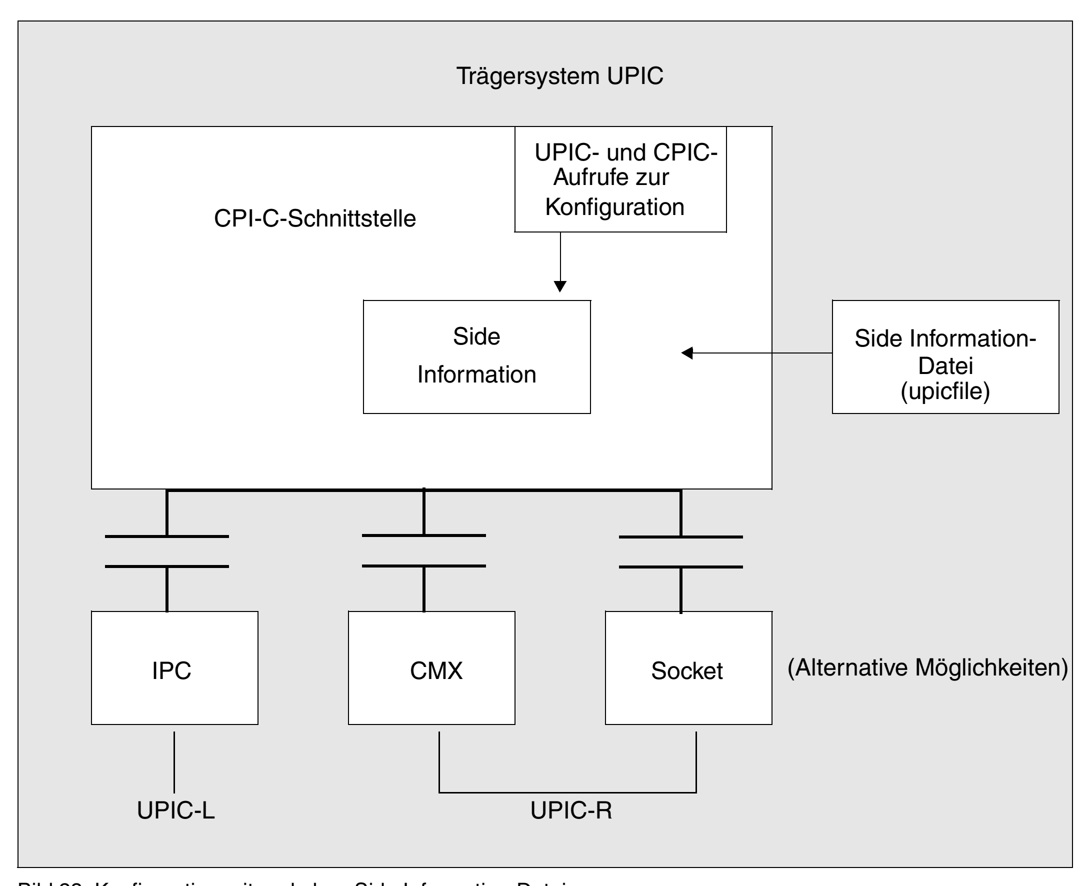


Bild 23: Konfiguration mit und ohne Side Information-Datei

6.1 Konfigurieren ohne upicfile

Zur Kommunikation zwischen UPIC und openUTM ist es erforderlich, dass sich sowohl der UPIC-Client als auch der UTM-Server lokal beim Kommunikationssystem mit einem Namen anmelden. UPIC meldet sich mit dem *local_name*, openUTM mit dem BCAMAPPL (Anwendungsname) beim Kommunikationssystem an. Eine Kommunikationsbeziehung zwischen Client und Server wird dadurch festgelegt, dass UPIC die UTM-Anwendung unter ihrem BCAMAPPL adressiert. openUTM erhält den lokalen Namen des Client, um den Client authentifizieren zu können (PTERM-Anweisung).

Wenn das Kommunikationssystem eine Rechner-übergreifende Kommunikation zulässt, dann muss der Client den Namen des fernen Rechners zur Adressierung hinzunehmen. Die vollständige Adresse des UTM-Partners besteht in diesem Fall aus BCAMAPPL und Rechnername.

UPIC adressiert die UTM-Anwendung über den *partner_LU_name*. Ein *partner_LU_name* wird als einstufig bezeichnet, wenn er nur die Adressierungsinformation über den lokalen Namen der UTM-Partner-Anwendung enthält. Der zweistufige *partner_LU_name* ist dadurch gekennzeichnet, dass er einen Punkt („.“) enthält. Der Teil links des Punktes ist der Anwendungsname, der Teil rechts des Punktes ist der Rechnername. Der Punkt selbst gehört nicht zur Adressierung.

Aus dem *partner_LU_name* werden die Werte für TSEL und HOSTNAME abgeleitet. Der linke Teil bis zum Punkt („.“), d.h. der Anwendungsname, wird dem TSEL zugeordnet. Der Teil rechts des Punktes, d.h. der Rechnername, wird dem HOSTNAME zugeordnet.

Adressierungskomponenten

- *local_name*

  Der *local_name* wird mit dem Aufruf *Enable_UTM_UPIC* gesetzt. Wenn ein leerer *local_name* (8 Leerzeichen und/oder Länge=0) bei diesem Aufruf übergeben wird, so wird ein vorbelegter *local_name* verwandt. Der *local_name* ist vorbelegt mit



  - `UPICL` bei UPIC-L
  - `UPICR` bei UPIC-R

  Er wird mit dem Aufruf *Specify_Local_Tsel* überschrieben.

Vergleich `upicfile`

Der Wert des *local_name* kann mit Hilfe einer `upicfile` überschrieben werden. Die `upicfile` ist in Abschnitt „Die Side Information-Datei (upicfile)“ auf Seite 297 beschrieben.

- *partner_LU_name*

  Der *partner_LU_name* ist nach dem *Initialize_Conversation*-Aufruf vorbelegt mit dem Wert:



  – `UTM` bei UPIC-L
  – `UTM.local` bei UPIC-R

  Er wird mit dem Aufruf *Set_Partner_LU_Name* überschrieben.

  Vergleich `upicfile`

  Der Wert des *partner_LU_name* kann auch mit Hilfe einer `upicfile` überschrieben werden. In der `upicfile` wird der *partner_LU_name* seinerseits über den *Symbolic Destination Name* adressiert.
  Die `upicfile` ist in Abschnitt „Die Side Information-Datei (upicfile)“ auf Seite 297 beschrieben.

- *Symbolic Destination Name*

  Der *Symbolic Destination Name* ist genau 8 Zeichen lang und wird beim *Initialize_Conversation*-Aufruf übergeben. Ein leerer *Symbolic Destination Name* besteht aus genau 8 Leerzeichen.
  Als *Symbolic Destination Name* **muss** ein leerer *Symbolic Destination Name* beim *Initialize_Conversation*-Aufruf übergeben werden.

  Vergleich `upicfile`

  Bei Verwendung einer `upicfile`, kann ein leerer *Symbolic Destination Name* beim *Initialize_Conversation*-Aufruf übergeben werden.
  Die `upicfile` ist in Abschnitt „Die Side Information-Datei (upicfile)“ auf Seite 297 beschrieben.

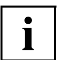
6.1.1 Konfiguration UPIC-L

X/W UPIC-L benutzt die Mechanismen der Interprozesskommunikation auf Windows- und Unix-Systemen. Bei diesen Kommunikationssystemen können der *local_name* und der *partner_LU_name* direkt auf die Adressierungsformate des Kommunikationssystems abgebildet werden. Sie müssen beachten, dass der *partner_LU_name* immer nur einstufig angegeben werden darf, da, bedingt durch das verwendete Kommunikationssystem, der UPIC-L Client und die UTM-Partner-Anwendung immer auf dem gleichen Rechner laufen. Die Angabe eines zweistufigen *partner_LU_name* enthielte auch eine Rechneradressierung. Da sie nie verwendet werden kann, wird ein zweistufiger *partner_LU_name* als Fehler behandelt.

6.1.2 Konfiguration UPIC-R

UPIC-R benutzt Transportsysteme zur Kommunikation. In der Praxis ist das in nahezu allen Fällen TCP/IP mit dem sogenannten RFC1006-Protokoll. Transportsysteme haben ihre eigenen Adressierungsvorschriften. Das RFC1006-Protokoll zeichnet sich dadurch aus, dass sich jede Transportsystem-Anwendung mit einem Namen beim Transportsystem anmeldet, dem Transport-Selektor (T-SEL). Die Partner adressieren einander über diese Namen. Da RFC1006 auf TCP/IP aufsetzt, werden auch folgende Adressierungsinformationen von TCP/IP benötigt:

- Rechnername
- Portnummer



Für BS2000 existiert die Vereinbarung, soweit als möglich die Portnummer 102 zu benutzen.

Für Unix- und Windows-Systeme gibt es keine allgemeine Empfehlung für eine Portnummer, die Portnummer 102 sollte aber nur mit Vorsicht verwendet werden.

Die Konfiguration von UPIC-R erfolgt über *local_name* und *partner_LU_name,* wobei der *local_name* auf den lokalen T-SEL abgebildet wird. Der Anwendungsname aus dem zweistufigen *partner_LU_name* wird auf den fernen T-SEL abgebildet, der Rechnername aus dem zweistufigen *partner_LU_name* ist der Name des Rechners im Netz. Der *partner_LU_name* **muss** zweistufig sein, da das beschriebene Verfahren sonst nicht funktioniert.

Bei der Abbildung des *local_name* und des Anwendungsnamen auf den T-SEL ist zu beachten, dass der Zeichencode des T-SEL nicht a priori festgelegt ist. Die beiden Rechner, auf denen Server und Client ablaufen, können zur Darstellung der T-SEL unterschiedliche Zeichencodes benützen (z.B. benutzen Windows-Systeme einen erweiterten ASCII-Zeichencode, BS2000/OSD den EBCDIC-Zeichencode). Daher muss das Format der Namen festgelegt werden. Zwischen UPIC und openUTM sind 3 Zeichenformate möglich: ASCII, EBCDIC und TRANSDATA. Der TRANSDATA Zeichensatz ist eine eingeschränkte Teilmenge des EBCDIC-Zeichensatzes. UPIC-R prüft, ob der von *local_name* und/oder der vom Anwendungsnamen verwendete Zeichensatz in den TRANSDATA-Zeichensatz umgewandelt werden kann. Ist das der Fall, wird das TRANSDATA-Zeichenformat verwendet, ansonsten wird das EBCDIC-Zeichenformat verwendet.

Sowohl dem *local_name* als auch dem *partner_LU_name* ist jeweils eine Portnummer zugeordnet. Die beiden Portnummern werden nicht aus den Namen abgeleitet, sie sind aber immer mit dem Wert 102 vorbelegt.

Dem *local_name* ist die lokale Portnummer zugeordnet. Der vorbelegte Wert kann überschrieben werden. Die lokale Portnummer ist ein rein formaler Wert, der keinerlei Wirkung hat und dessen Angabe nur aus Gründen der Kompatibilität gepflegt wird. Bei der Konfiguration von UPIC-R sollte er vernachlässigt werden.

Dem *partner_LU_name* ist die ferne Portnummer zugeordnet. Der fernen Portnummer kommt im Gegensatz zur lokalen Portnummer eine wesentliche Bedeutung zu, da über sie die UTM-Partner-Anwendung adressiert wird. In der Praxis genügt es in den allermeisten Fällen, den vorbelegten Wert 102 zu verwenden. BCAM und CMX unterstützen immer den Port 102 als zentralen Zugangsport für RFC1006. Die Wahl eines anderen Port ist zwar möglich, sie erfordert aber auf der Server-Seite einen erhöhten Konfigurationsaufwand, z.B. müssen dann für BS2000/OSD-System BCMAP Einträge erstellt werden. Solche Konfigurationen setzen eine gewisse Erfahrung voraus und werden hier nicht beschrieben. Wenn die UTM-Partner-Anwendung auf einem System läuft, das PCMX als Zugang zum Transportsystem nutzt, dann kann der Port 102 im allgemeinen nicht verwendet werden. Dann muss der Wert der fernen Portnummer mit dem Wert überschrieben werden, der von der UTM-Anwendung genutzt wird.

Die Werte T-SEL, T-SEL-Format und lokale Portnummer des *local_name* können mit folgenden Aufrufen überschrieben werden:

Specify_Local_Tsel
Specify_Local_Tsel_Format und
Specify_Local_Port

Die Werte können auch durch Einträge in der `upicfile` überschrieben werden. Die jeweiligen Werte werden dabei über Schlüsselwörter festgelegt. Die `upicfile` ist in Abschnitt „Die Side Information-Datei (upicfile)“ auf Seite 297 beschrieben.

Um die Adressierungsinformationen für das Netzwerk zu bilden, genügt es, den *local_name* anzugeben und mittels der internen Regeln von UPIC die Netzwerkadressierung erstellen zu lassen. Es ist auch zulässig und vorgesehen, einen oder mehrere der aus dem *local_name* abgeleiteten Werte mit den angegebenen Aufrufen zu überschreiben. Dabei ist jede Mischung aus abgeleiteten bzw. vorbelegten und explizit gesetzten Werten zulässig. Ebenso ist es zulässig, alle aus dem *local_name* abgeleiteten Werte zu überschreiben. Wenn Sie diese Art der Konfigurierung wählen, ist der *local_name* belanglos. Sie können dann jeden beliebigen *local_name* angeben, wenn er nur die formalen Kriterien des *Enable_UTM_UPIC*-Aufrufs einhält.

Die Werte Rechnername (bzw. die daraus abgeleitete Internet-Adresse), T-SEL, T-SEL-Format und ferne Portnummer können mit folgenden Aufrufen überschrieben werden:

Set_Partner_Host_Name
Set_Partner_IP_Address
Set_Partner_Tsel
Set_Partner_Tsel_Format
Set_Partner_Port

Wenn die Aufrufe *Set_Partner_Host_Name* und *Set_Partner_IP_Address* beide aufgerufen werden, wird der Aufruf *Set_Partner_Host_Name* ignoriert. Die Werte können auch durch Einträge in der `upicfile` überschrieben werden. Die jeweiligen Werte werden dabei über Schlüsselwörter festgelegt. Die `upicfile` ist in Abschnitt „Die Side Information-Datei (upicfile)“ auf Seite 297 beschrieben.

Um die Adressierungsinformationen für das Netzwerk zu bilden genügt es vielfach, den *partner_LU_name* anzugeben und mittels der internen Regeln von UPIC die Netzadressierung erstellen zu lassen. Es ist auch zulässig und vorgesehen, einen oder mehrere der aus dem *partner_LU_name* abgeleiteten Werte mit den angegebenen Aufrufen zu überschreiben. Dabei ist jede Mischung aus abgeleiteten bzw. vorbelegten und explizit gesetzten Werten zulässig. Ebenso ist es zulässig, alle aus dem *partner_LU_name* abgeleiteten Werte zu überschreiben. Wenn Sie diese Art der Konfiguration wählen, wird der *partner_LU_name* belanglos. Sie können einen beliebigen *partner_LU_name* angeben, wenn er nur die formalen Kriterien erfüllt, die an ihn gestellt werden (er muss unter anderem immer zweistufig sein).

6.1.3 Konfiguration mit TNS-Einträgen

X/W Wenn UPIC-R zur Kommunikation die Transportsystemkomponente PCMX benutzt, dann kann die Konfiguration auch durch TNS-Einträge erfolgen. UPIC-R mit PCMX versucht immer zuerst, zum *local_name* und zum *partner_LU_name* einen globalen Namen im TNS-Directory zu finden. Wenn ein globaler Name zu dem *local_name* und oder zu dem *partner_LU_name* gefunden wird, dann wird er auch verwendet. Alle sonstigen Konfigurationseinstellungen werden ignoriert. Wird jedoch kein TNS-Eintrag zum *local_name* und/oder zum *partner_LU_name* gefunden, dann erfolgt die Konfiguration wie oben beschrieben.

6.1.4 Konfiguration mit BCMAP-Einträgen

B Wenn UPIC auf BS2000/OSD zur Kommunikation die Transportsystemkomponente CMX(BS2000) benutzt, dann wird die Konfiguration durch BCMAP-Einträge beeinflusst.

B BCMAP-Einträge für die Client-Anwendung und für die UTM-Partner-Anwendung sind nur in wenigen Ausnahmefällen nötig, wenn die Kommunikation mit einer UTM-Anwendung auf Windows-Systemen erfolgt.

B Die Wirkung von BCMAP-Einträgen kann vom UPIC-Client nicht beeinflusst werden.

B BCMAP-Einträge können sowohl für den *local_name* als auch für den *partner_LU_name* erstellt werden. BCMAP-Einträge für den *local_name* werden nicht empfohlen.

B BCMAP-Einträge für den *partner_LU_name* sind im Allgemeinen erforderlich, wenn ein UPIC -Client auf BS2000/OSD mit einer UTM-Anwendung auf Windows-Systemen kommunizieren will.

6.2 Die Side Information-Datei (upicfile)

Die `upicfile` müssen Sie selbst erstellen. Sie hat folgendes Format:



- In Unix- und Windows-Systemen muss diese Datei eine reine Textdatei sein mit dem Namen `upicfile`. Wenn Sie einen anderen Dateinamen wählen, müssen Sie die Umgebungsvariable UPICFILE entsprechend setzen.
- Im BS2000 müssen Sie eine SAM-Datei erstellen mit dem Namen `upicfile`. Wenn Sie einen anderen Dateinamen wählen, müssen Sie die Jobvariable UPICFILE entsprechend setzen.

Diese Datei wird von allen Client-Programmen verwendet, z.B. bei den Aufrufen *Enable_UTM_UPIC* oder *Initialize_Conversation*.

Der Zugriff erfolgt mit der Umgebungsvariablen bzw. Jobvariablen UPICPATH. Damit kann das Verzeichnis bestimmt werden, in dem die Datei steht. Ist diese Variable nicht gesetzt, wird die Datei im aktuellen Verzeichnis gesucht.

Die `upicfile` kennt folgende Arten von Einträgen:

- Side Information Einträge für die Kommunikationspartner, die im Client-Programm über den Symbolic Destination Name adressiert werden.
- Side Information Einträge für die Kommunikationspartner in einem openUTM-Cluster, die im Client-Programm über den Symbolic Destination Name adressiert werden.
- Side Information Einträge für die lokale Anwendung, die im Client-Programm über den lokalen Anwendungsnamen adressiert werden. Diese Einträge sind optional.

Um das Layout der `upicfile` lesbarer zu gestalten, ist es erlaubt, dass die Datei auch Leer- bzw. Kommentarzeilen enthält. Kommentarzeilen sind dadurch gekennzeichnet, dass sie mit einem „*“-Zeichen in Spalte 1 beginnen. Dabei ist zu beachten, dass ein Semikolon immer als Zeilenabschluss interpretiert wird, auch innerhalb einer Kommentarzeile.

6.2.1 Side Information für stand-alone UTM-Anwendungen

Jeder Kommunikationspartner wird im Client-Programm durch seinen Symbolic Destination Name adressiert. Dieser Name wird beim Initialisieren einer Conversation (Aufruf *Initialize_Conversation*) angegeben.

Für jeden *Symbolic Destination Name*, der im Programm verwendet wird, muss in der `upicfile` ein Eintrag erstellt werden. Jeder Eintrag belegt eine Zeile in der `upicfile`.

Der Eintrag hat für stand-alone UTM-Anwendungen folgende Form:

SD oder HD	symbolic destination name	blank	partner _LU_ name	blank	transaction-code	blank	Schlüssel-wörter	Zeilen-abschluss zeichen
2 Byte	8 Byte	1 Byte	1-32 Byte[1]	1 Byte	1-8 Byte	1 Byte		
				optional		optional		


[1] Bei lokaler Anbindung mit UPIC-Local darf „partner_LU_ name“ nur bis zu 8 Bytes lang sein.

Beschreibung des Eintrags

- Die Namen, die im Eintrag angegeben werden, müssen durch Blanks voneinander getrennt werden.
  Ausnahme:
  Zwischen dem Kennzeichen SD/HD und dem Symbolic Destination Name darf kein Blank stehen.
- Kennzeichen SD/HD:
  Die Zeile beginnt mit dem Kennzeichen SD oder HD. Das Kennzeichen gibt an, ob UPIC beim Senden und Empfangen von Daten eine automatische Code-Konvertierung durchführen soll oder nicht. Zur Code-Konvertierung siehe auch Abschnitt „Code-Konvertierung“ auf Seite 68.

X/W Windows-, Unix-Systeme:

X/W Geben Sie HD an, dann wird beim Senden und beim Empfangen eine automatische Code-Konvertierung der Benutzerdaten durchgeführt.
Daten, die an die UTM-Partner-Anwendung gesendet werden, werden vom lokal verwendeten Code nach EBCDIC konvertiert.
Daten, die von der Partner-Anwendung eintreffen, werden von EBCDIC in den lokalen Code konvertiert.

X/W Geben Sie SD an, dann wird keine automatische Code-Konvertierung durchgeführt.

B B
BS2000/OSD:

In BS2000/OSD haben die Kennzeichen die umgekehrte Bedeutung.

B B B B
HD bedeutet in UPIC auf BS2000/OSD, dass beim Senden und Empfangen von Daten im lokalen System keine automatische Code-Konvertierung durchgeführt wird. HD sollte immer angegeben werden, wenn der Client mit einer UTM-Anwendung auf BS2000/OSD kommuniziert (BS2000/OSD - BS2000/OSD-Kopplung).

B B B B
SD bedeutet, dass vor dem Senden von Daten eine EBCDIC->ASCII Konvertierung durchgeführt wird und beim Empfangen eine ASCII->EBCDIC Konvertierung.
SD sollte nur für Verbindungen zu UTM-Anwendungen auf Unix- oder WIndows-Systemen angegeben werden.

Das Kennzeichen SD/HD in der upicfile kann mit dem *Set_Convertion*-Aufruf überschrieben werden.

- symbolic destination name
  Der Symbolic Destination Name muss genau acht Zeichen lang sein.

- partner_LU_name
  Der *partner_LU_name* kann bei Kopplungen über UPIC-Remote zwischen 1 und 32 Zeichen lang sein. Für *partner_LU_name* ist der symbolische Name anzugeben, mit dem die UTM-Partner-Anwendung dem Kommunikationssystem bekannt ist.
  Bei Kopplungen über UPIC-Remote sollten Sie den *partner_LU_name* immer zweistufig in der Form *applicationname.processorname* (getrennt durch einen Punkt) angeben. Aus dem zweistufigen *partner_LU_name* werden die Werte für TSEL (=*applicationname*) und HOSTNAME (=*processorname*) abgeleitet.

B B
Im BS2000 müssen Sie den *partner_LU_name* zweistufig angeben. *processorname* muss dann mit dem Namen des fernen Rechners im BCAM-RDF übereinstimmen.

Beispiel

Angabe in der `upicfile`: `SDsymbdest UTMAPPL1.D123ZE45`

Ein Eintrag in der upicfile kann mit dem *Set_Partner_LU_Name*-Aufruf überschrieben werden.

Die einzelnen Werte eines zweistufigen *partner_LU_name* können mit Einträgen in der side information datei (HOSTNAME=, TSEL=) oder mit den Aufrufen *Set_Partner_Hostname* und *Set_Partner_Tsel* überschrieben werden.

X/W X/W X/W
UPIC-L:
Bei der lokalen Anbindung an eine UTM-Anwendung mit UPIC-L darf der Partnername nur bis zu acht Zeichen lang sein. Die Angabe muss einstufig erfolgen.

- transactioncode (Angabe optional):
  Es kann der Transaktionscode eines UTM-Services angegeben werden. Der Transaktionscode ist ein bis zu 8 Zeichen langer Name. Der angegebene Transaktionscode muss in der UTM-Partner-Anwendung generiert (TAC-Anweisung) oder dynamisch konfiguriert worden sein.
  Die Angabe eines Transaktionscodes in einem Eintrag ist optional. Fehlt die Angabe, so muss der Transaktionscode (Name des Services) im Programm mit dem *Set_TP_Name*-Aufruf angegeben werden.

  Ein Eintrag in der `upicfile` kann mit dem *Set_TP_Name*-Aufruf überschrieben werden.

- Schlüsselwörter (alle Angaben optional)
  Mit folgenden Schlüsselwörtern können Sie die UPIC-spezifischen conversation characteristics (siehe hierzu auch „Conversation Characteristics“ auf Seite 51) in der `upicfile` beeinflussen. Mit den Schlüsselwörtern geben Sie die Adressierungsinformationen an und legen fest, ob verschlüsselt werden soll.
  Sie können die Schlüsselwörter nach dem Partnernamen oder nach dem Transaktionscode jeweils getrennt durch ein Leerzeichen angeben. Die Reihenfolge und Anzahl der Schlüsselwörter ist beliebig. Mehrere Schlüsselwörter werden durch Leerzeichen getrennt.

  ENCRYPTION-LEVEL={NONE | 0 | 1 | 2 | 3 | 4}

  Mit ENCRYPTION-LEVEL legen Sie fest, ob die Daten für die Conversation verschlüsselt werden sollen oder nicht und welche Verschlüsselungsebene verwendet werden soll.

  Geben Sie ENCRYPTION-LEVEL=NONE oder ENCRYPTION-LEVEL=0 an (beides hat die gleiche Wirkung), so werden die Benutzerdaten nicht verschlüsselt. Verlangt jedoch die UTM-Anwendung auf einer Verbindung die Verschlüsselung der Daten, wird die Verschlüsselungsebene automatisch hochgesetzt. Dasselbe geschieht, wenn UPIC auf einer Verbindung mit ENCRYPTION-LEVEL=NONE einen TAC aufruft, der mit Verschlüsselung generiert ist und UPIC keine Benutzerdaten beim Aufruf des TACs mitsendet. Durch den Empfang verschlüsselter Daten setzt UPIC den Wert für die Verschlüsselungsebene automatisch hoch.

  Wenn Sie ENCRYPTION-LEVEL=1, 2, 3 oder 4 angeben und openUTM auf der Verbindung entsprechend verschlüsseln kann, dann werden alle Benutzerdaten der folgenden Conversation mit derselben Ebene verschlüsselt übertragen.

  Die Werte1 bis 4 bedeuten:

  1. Verschlüsseln der Benutzerdaten mit dem DES-Algorithmus. Für den Austausch des DES-Schlüssels wird ein RSA-Schlüssel mit einer Schlüssellänge von 200 bit verwendet.
  2. Verschlüsseln der Benutzerdaten mit dem AES-Algorithmus. Für den Austausch des AES-Schlüssels wird ein RSA-Schlüssel mit einer Schlüssellänge von 512 bit verwendet.

3 Verschlüsseln der Benutzerdaten mit dem AES-Algorithmus. Für den Austausch des AES-Schlüssels wird ein RSA-Schlüssel mit einer Schlüssellänge von 1024 bit verwendet.

4 Verschlüsseln der Benutzerdaten mit dem AES-Algorithmus. Für den Austausch des AES-Schlüssels wird ein RSA-Schlüssel mit einer Schlüssellänge von 2048 bit verwendet.

Unterstützt openUTM die angegebene Verschlüsselungsebene nicht, dann wird die Conversation beendet.

Der Wert wird ignoriert, wenn eine UTM-Anwendung nicht verschlüsseln kann, weil
- openUTM-Crypt nicht installiert ist
- sie nicht verschlüsseln will, da der Client-Partner als vertrauenswürdig (trusted) generiert wurde

UPIC-L:
Der Wert für ENCRYPTION-LEVEL wird ignoriert. Der Eintrag in der `upicfile` kann mit dem *Set_Conversation_Encryption_Level*-Aufruf überschrieben werden.

HOSTNAME=*hostname*
Der Hostname ist der Prozessorname und kann bis zu 32 Zeichen lang sein. Der Hostname überschreibt den beim *Initialize_Conversation* zugewiesenen Wert.

Ein Eintrag in der `upicfile` kann mit dem *Set_Partner_Host_Name*-Aufruf überschrieben werden.



UPIC-L:
Der Wert für HOSTNAME wird ignoriert.

IP-ADDRESS=*nnn.nnn.nnn.nnn* (IPv4) bzw. = *x: x: x: x: x: x: x: x* (IPv6).
Es kann eine Internet-Adresse im Format IPv4 und IPv6 angegeben werden:

- Wird die Internet-Adresse in der üblichen Punktnotation angegeben, dann wird sie als IPv4-Adresse interpretiert.
- Wird die Internet-Adresse in der Form *x: x: x: x: x: x: x: x* angegeben, dann wird sie als IPv6-Adresse interpretiert. Dabei ist *x* eine hexadezimale Zahl zwischen 0 und FFFF. Die alternativen Schreibweisen von IPv6-Adressen (z.B. Weglassen von Nullen durch :: oder IPv6 mapped format) sind erlaubt.

Wenn eine Internet-Adresse angegeben wird, wird der Wert von HOSTNAME ignoriert. Ein Eintrag in der `upicfile` kann mit dem *Set_Partner_IP_Address*-Aufruf überschrieben werden.



UPIC-L:
Der Wert für IP-ADDRESS wird ignoriert.

B
B

UPIC auf BS2000/OSD mit CMX als Kommunikationssystem
Der Wert für IP-ADDRESS wird ignoriert.

PORT=*listener-port*

Die Portnummer wird nur für das Adressformat RFC1006 angegeben. Die Portnummer kann einen Wert zwischen 0 bis 32767 annehmen. Diese Portnummer überschreibt den Wert für die Portnummer, der beim *Initialize_Conversation* zugewiesen wurde. Die Angabe von PORT ist optional.

Wenn für diesen Kommunikationspartner TNS-freier Betrieb festgelegt ist, wird statt 102 der Wert von PORT als Portnummer benutzt.

Ein Eintrag in der `upicfile` kann mit dem *Set_Partner_Port*-Aufruf überschrieben werden.

X/W UPIC-L Der Wert für PORT wird ignoriert.

B UPIC auf BS2000/OSD mit CMX als Kommunikationssystem
B Der Wert für PORT wird ignoriert.

PROTOCOL= {34 | 40}

In PROTOCOL legen Sie fest, ob über das UPIC-Protokoll der Version V4.0 (PROTOCOL=40) oder über das UPIC-Protokoll der Version V3.4 (PROTOCOL=34) kommuniziert werden soll. Die Angabe von PROTOCOL ist optional.

PROTOCOL=34 ist für Conversations mit UTM-Anwendungen < V4.0 notwendig.

Geben Sie PROTOCOL nicht an, dann versucht UPIC-R mit CMX zunächst, eine Conversation auf Basis des erweiterten Protokolls (40) aufzubauen. Falls dies misslingt, versucht UPIC als nächstes, die Conversation auf Basis des Protokolls der V3.4 (34) aufzubauen. UPIC-R mit Socket als Kommunikationssystem benutzt diesen Automatismus nicht.

Ist der Conversation-Aufbau gemäß dem Protokoll der V3.4 erfolgt, so können z.B. die UPIC-Funktionen „Formatdaten austauschen“ und „Funktionstasten betätigen“ nicht genutzt werden.

Geben Sie PROTOCOL an, dann versucht UPIC den Conversation-Aufbau nur gemäß dem angegebenen Protokoll. Geben Sie PROTOCOL=40 an und schlägt der Verbindungsversuch fehl, weil der UTM-Server die Version des UPIC-Protokolls nicht unterstützt, dann versucht UPIC <u>nicht</u> mehr die Conversation auf Basis des Protokolls der Version V3.4 aufzubauen.

Ein UTM-Server ab der Version V4.0 erkennt beim Aufbau der Conversation, welches Protokoll der Client unterstützt.

Ein Eintrag in der upicfile kann mit dem *Set_Communication_Protocol*-Aufruf überschrieben werden.

X/W UPIC-L Der Wert für PROTOCOL wird ignoriert.

RSA-KEY=*rsa-key*

Es kann der öffentliche Teil des RSA-Schlüssels der Partner-Anwendung angegeben werden. Wenn der öffentliche Schlüssel angegeben ist, vergleicht die UPIC-Bibliothek den angegebenen Schlüssel mit dem, den sie von der UTM-Partner-Anwendung beim Verbindungsaufbau erhält. Unterscheiden sich beide Schlüssel in mindestens einem Byte oder auch nur in der Länge, so wird die Verbindung von der UPIC-Bibliothek sofort wieder abgebaut. Mit diesem Verfahren kann die Echtheit des Schlüssels überprüft werden.

X/W UPIC-L Der Wert für RSA-KEY wird ignoriert.

T-SEL=*transport-selektor*

Der Transport-Selektor (T-SEL) der Transportadresse adressiert die Partner-Anwendung innerhalb des fernen Systems. Er muss mit den Angaben im fernen System übereinstimmen. Der Transport-Selektor ist ein bis zu 8 Zeichen langer Name. Der angegebene T-SEL überschreibt den beim *Initialize_Conversation* zugewiesenen Wert. Die Angabe von T-SEL ist optional.

Der Eintrag in der `upicfile` kann mit dem *Set_Partner_Tsel*-Aufruf überschrieben werden.

X/W UPIC-L Der Wert für T-SEL wird ignoriert.

T-SEL-FORMAT={T | E | A }

T-SEL-FORMAT ist der Formatindikator des Transport-Selektors. Gültige Formate sind

T für TRANSDATA
E für EBCDIC
A für ASCII

T-SEL-FORMAT überschreibt den beim *Initialize_Conversation* zugewiesenen Wert. Die Angabe von T-SEL-FORMAT ist optional.

Wenn für einen Kommunikationspartner TNS-freier Betrieb festgelegt ist, wird der Wert von TSEL-FORMAT benutzt. Der Eintrag in der `upicfile` kann mit dem *Set_Partner_Tsel_Format*-Aufruf überschrieben werden.

X/W UPIC-L Der Wert für T-SEL-FORMAT wird ignoriert.

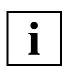
- Zeilenabschlusszeichen:
  Das Zeichen, das den Eintrag abschließt, ist für die verschiedenen Plattformen, für die die `upicfile` erstellt wird, unterschiedlich:

W W W
  – *Windows-Systeme:*
  Eine Zeile wird durch Carriage Return und Line Feed (Return-Taste) abgeschlossen. Ein Semikolon vor dem Carriage Return-Zeichen ist optional.

– *Unix-Systeme:*
  Die Zeile wird mit einem <newline>-Zeichen (Line Feed) abgeschlossen. Ein Semikolon vor dem <newline>-Zeichen ist optional.

– *BS2000/OSD:*
  Das Zeilenende wird durch ein Semikolon (;) dargestellt. Danach darf kein Leerzeichen mehr folgen.

Falls in einer Zeile (Inhalt des Side Information Eintrags) ein Semikolon steht, reagiert UPIC so, als ob die Zeile dort abgeschlossen wäre und interpretiert den Rest der Zeile als neue Zeile (bis zum nächsten Zeilenabschlusszeichen).



Beachten Sie, dass im BS2000/OSD das nächste Zeilenabschlusszeichen auch wieder ein Semikolon ist. BS2000-Editoren, z.B. EDT haben eine andere Sicht auf Zeilen als UPIC. Wenn nach dem Semikolon der Zeile *n* im Editor noch ein Blank folgt und die Zeile *n*+1 beginnt mit SD und endet mit einem Semikolon, dann sieht UPIC eine Zeile, die mit „ SD“ beginnt und **nicht** mit „SD“.
Der „Symbolic Destination Name“ in dieser Zeile wird nicht gefunden.

DEFAULT-Server definieren

Sie können für Ihre Client-Anwendung einen DEFAULT-Server bzw. einen DEFAULT-Service definieren (siehe auch Abschnitt „DEFAULT-Server und DEFAULT-Name eines Client“ auf Seite 94). Ein Client-Programm wird mit dem DEFAULT-Server/Service verbunden, wenn im Programm als Symbolic Destination Name ein leerer Name übergeben wird. Im DEFAULT-Eintrag geben Sie statt des Symbolic Destination Name den Wert `.DEFAULT` an. Der DEFAULT-Server-Eintrag muss also folgendes Format haben.

SD oder HD	.DEFAULT	blank	partner _LU_ name	blank	transaction-code	blank	Schlüssel-wörter	Zeilen-abschluss zeichen
2 Byte		1 Byte	1-32 Byte[1]	1 Byte	1-8 Byte	1 Byte		
				optional		*optional*		

[1] Bei lokaler Anbindung mit UPIC-Local darf „partner_LU_name“ nur bis zu 8 Bytes lang sein.

Mit einem solchen Eintrag definieren Sie die UTM-Partner-Anwendung *partner_LU_name* als DEFAULT-Server. Geben Sie einen Transaktionscode an, dann definieren Sie darüber hinaus den zugehörigen Service als DEFAULT-Service. Einen anderen Service am DEFAULT-Server rufen Sie auf, wenn Sie im Programm mit dem Aufruf *Set_TP_Name* einen anderen Transaktionscode setzen (z.B. KDCDISP für den Vorgangs-Wiederanlauf). Die Angabe in *Set_TP_Name* überschreibt den Wert von *transactioncode* im Side Information Eintrag.

6.2.2 Side Information für UTM-Cluster-Anwendungen

Jeder Kommunikationspartner, also auch eine UTM-Cluster-Anwendung, wird im Client-Programm durch seinen Symbolic Destination Name adressiert. Dieser Name wird beim Initialisieren einer Conversation (Aufruf *Initialize_Conversation*) angegeben. Für jeden *Symbolic Destination Name*, der im Programm verwendet wird, müssen Sie in der `upicfile`-Einträge erstellen.

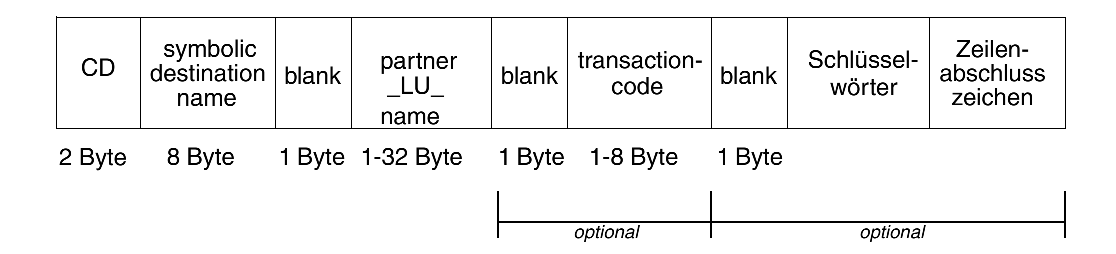
Eine UTM-Cluster-Anwendung besteht aus mehreren identischen Knoten-Anwendungen, die auf den einzelnen Knoten des Clusters ablaufen. Damit ein UPIC-Client alle Knoten-Anwendungen der UTM-Cluster-Anwendung auf einfache Weise erreichen kann, müssen Sie in der `upicfile` einen openUTM-Cluster konfigurieren. Dabei müssen Sie folgende Regeln beachten.

Regeln bei der Konfiguration einer openUTM-Cluster-Anwendung

- Zu einem *Symbolic Destination Name* müssen Sie pro Knoten-Anwendung einen eigenen Eintrag in der `upicfile` mit Kennzeichen CD erstellen. Wenn die UTM-Cluster-Anwendung z.B. aus drei Knoten-Anwendungen besteht, dann müssen Sie drei Einträge mit demselben *Symbolic Destination Name* erstellen.
- Alle Einträge für einen bestimmten *Symbolic Destination Name* müssen direkt hintereinander stehen, siehe auch Beispiel auf Seite 310.
- Die Einträge für einen bestimmten *Symbolic Destination Name* unterscheiden sich nur in den Adressangaben der Knoten (*partner_LU_name* oder, falls verwendet, den Schlüsselwörtern HOSTNAME und IP-ADDRESS). Die Angaben für *transaktionscode* und die übrigen Schlüsselwörter müssen übereinstimmen.

Format eines Eintrags

Jeder Eintrag belegt eine Zeile in der `upicfile`. Ein Eintrag hat folgende Form:



CD	symbolic destination name	blank	partner _LU_ name	blank	transaction-code	blank	Schlüssel-wörter	Zeilen-abschluss zeichen
2 Byte	8 Byte	1 Byte	1-32 Byte	1 Byte	1-8 Byte	1 Byte		
				optional	optional	optional	optional	optional

Beschreibung des Eintrags

- Die Namen, die im Eintrag angegeben werden, müssen durch Blanks voneinander getrennt werden.
  Ausnahme:
  Zwischen dem Kennzeichen CD und dem Symbolic Destination Name darf kein Blank stehen.

- Kennzeichen CD:
  Die Zeile beginnt mit dem Kennzeichen CD. Das Kennzeichen hat keine Auswirkung auf die automatische Code-Konvertierung (siehe auch „CONVERTION={IMPLICIT | NO}“ auf Seite 309).

- symbolic destination name
  Der Symbolic Destination Name muss genau acht Zeichen lang sein.

  Die Kombination `CD`*symbolic_destination_name* darf in der `upicfile` beliebig oft vorkommen.

- partner_LU_name
  Der *partner_LU_name* kann zwischen 1 und 32 Zeichen lang sein.
  Für *partner_LU_name* ist der symbolische Name anzugeben, unter dem die UTM-Partner-Anwendung dem Kommunikationssystem bekannt ist.

  Sie sollten den *partner_LU_name* immer zweistufig in der Form *applicationname.processorname* (getrennt durch einen Punkt) angeben. Aus dem zweistufigen *partner_LU_name* werden die Werte für TSEL (=*applicationname*) und HOSTNAME (=*processorname*) abgeleitet.

  B
  B
  Im BS2000 müssen Sie den *partner_LU_name* zweistufig angeben. *processorname* muss dann mit dem Namen des fernen Rechners im BCAM-RDF übereinstimmen.

  Beispiel
  Angabe in der `upicfile`: `CDsymbdest UTMAPPL1.D123ZE45`

  Ein Eintrag in der upicfile kann **nicht** mit dem *Set_Partner_LU_Name*-Aufruf überschrieben werden. Die einzelnen Werte eines zweistufigen *partner_LU_name* dürfen im Programm nicht überschrieben werden, ein entsprechender Aufruf wird abgelehnt.

- transactioncode (Angabe optional):
  Es kann der Transaktionscode eines UTM-Services angegeben werden. Der Transaktionscode ist ein bis zu 8 Zeichen langer Name. Der angegebene Transaktionscode muss in der UTM-Partner-Anwendung generiert (TAC-Anweisung) oder dynamisch konfiguriert worden sein.
  Die Angabe eines Transaktionscodes in einem Eintrag ist optional. Fehlt die Angabe, so muss der Transaktionscode (Name des Services) im Programm mit dem *Set_TP_Name*-Aufruf angegeben werden.

  Ein Eintrag in der `upicfile` kann mit dem *Set_TP_Name*-Aufruf überschrieben werden.

- Schlüsselwörter (alle Angaben optional)
  Mit folgenden Schlüsselwörtern können Sie die UPIC-spezifischen conversation characteristics (siehe hierzu auch „Conversation Characteristics“ auf Seite 51) in der `upicfile` beeinflussen. Mit den Schlüsselwörtern geben Sie die Adressierungsinformationen an und legen fest, ob verschlüsselt werden soll.
  Sie können die Schlüsselwörter nach dem Partnernamen oder nach dem Transaktionscode jeweils getrennt durch ein Leerzeichen angeben. Die Reihenfolge und Anzahl der Schlüsselwörter ist beliebig. Mehrere Schlüsselwörter werden durch Leerzeichen getrennt.

  ENCRYPTION-LEVEL={NONE | 0 | 1 | 2 | 3 | 4}

  Mit ENCRYPTION-LEVEL legen Sie fest, ob die Daten für die Conversation verschlüsselt werden sollen oder nicht und welche Verschlüsselungsebene verwendet werden soll.

  Geben Sie ENCRYPTION-LEVEL=NONE oder ENCRYPTION-LEVEL=0 an (beides hat die gleiche Wirkung), so werden die Benutzerdaten nicht verschlüsselt. Verlangt jedoch die UTM-Anwendung auf einer Verbindung die Verschlüsselung der Daten, wird die Verschlüsselungsebene automatisch hochgesetzt. Dasselbe geschieht, wenn UPIC auf einer Verbindung mit ENCRYPTION-LEVEL=NONE einen TAC aufruft, der mit Verschlüsselung generiert ist und UPIC keine Benutzerdaten beim Aufruf des TACs mitsendet. Durch den Empfang verschlüsselter Daten setzt UPIC den Wert für die Verschlüsselungsebene automatisch hoch.

  Wenn Sie ENCRYPTION-LEVEL=1, 2, 3 oder 4 angeben und openUTM auf der Verbindung entsprechend verschlüsseln kann, dann werden alle Benutzerdaten der folgenden Conversation mit derselben Ebene verschlüsselt übertragen.

  Die Werte1 bis 4 bedeuten:

  1 Verschlüsseln der Benutzerdaten mit dem DES-Algorithmus. Für den Austausch des DES-Schlüssels wird ein RSA-Schlüssel mit einer Schlüssellänge von 200 bit verwendet.

  2 Verschlüsseln der Benutzerdaten mit dem AES-Algorithmus. Für den Austausch des AES-Schlüssels wird ein RSA-Schlüssel mit einer Schlüssellänge von 512 bit verwendet.

  3 Verschlüsseln der Benutzerdaten mit dem AES-Algorithmus. Für den Austausch des AES-Schlüssels wird ein RSA-Schlüssel mit einer Schlüssellänge von 1024 bit verwendet.

  4 Verschlüsseln der Benutzerdaten mit dem AES-Algorithmus. Für den Austausch des AES-Schlüssels wird ein RSA-Schlüssel mit einer Schlüssellänge von 2048 bit verwendet.

  Unterstützt openUTM die angegebene Verschlüsselungsebene nicht, dann wird die Conversation beendet.

Der Wert wird ignoriert, wenn eine UTM-Anwendung nicht verschlüsseln kann, weil

- openUTM-Crypt nicht installiert ist
- sie nicht verschlüsseln will, da der Client-Partner als vertrauenswürdig (trusted) generiert wurde

HOSTNAME=*hostname*

Der Hostname ist der Prozessorname und kann bis zu 32 Zeichen lang sein. Der Hostname überschreibt den beim *Initialize_Conversation* zugewiesenen Wert.

Ein Eintrag in der `upicfile` kann **nicht** mit dem *Set_Partner_Host_Name*-Aufruf überschrieben werden.

IP-ADDRESS=*nnn.nnn.nnn.nnn* (IPv4) bzw. = *x: x: x: x: x: x: x: x* (IPv6).

Es kann eine Internet-Adresse im Format IPv4 und IPv6 angegeben werden:

- Wird die Internet-Adresse in der üblichen Punktnotation angegeben, dann wird sie als IPv4-Adresse interpretiert.
- Wird die Internet-Adresse in der Form *x: x: x: x: x: x: x: x* angegeben, dann wird sie als IPv6-Adresse interpretiert. Dabei ist *x* eine hexadezimale Zahl zwischen 0 und FFFF. Die alternativen Schreibweisen von IPv6-Adressen (z.B. Weglassen von Nullen durch :: oder IPv6 mapped format) sind erlaubt.

Wenn eine Internet-Adresse angegeben wird, wird der Wert von HOSTNAME ignoriert. Ein Eintrag in der `upicfile` kann **nicht** mit dem *Set_Partner_IP_Address*-Aufruf überschrieben werden.

B UPIC auf BS2000/OSD mit CMX als Kommunikationssystem
B Der Wert für IP-ADDRESS wird ignoriert.

PORT=*listener-port*

Die Portnummer wird nur für das Adressformat RFC1006 angegeben. Die Portnummer kann einen Wert zwischen 0 bis 32767 annehmen. Diese Portnummer überschreibt den Wert für die Portnummer, der beim *Initialize_Conversation* zugewiesen wurde. Die Angabe von PORT ist optional.

Wenn für diesen Kommunikationspartner TNS-freier Betrieb festgelegt ist, wird statt 102 der Wert von PORT als Portnummer benutzt.

Ein Eintrag in der `upicfile` kann mit dem *Set_Partner_Port*-Aufruf überschrieben werden.

B UPIC auf BS2000/OSD mit CMX als Kommunikationssystem
B Der Wert für PORT wird ignoriert.

- RSA-KEY=*rsa-key*
  Es kann der öffentliche Teil des RSA-Schlüssels der Partner-Anwendung angegeben werden. Wenn der öffentliche Schlüssel angegeben ist, vergleicht die UPIC-Bibliothek den angegebenen Schlüssel mit dem, den sie von der UTM-Partner-Anwendung beim Verbindungsaufbau erhält. Unterscheiden sich beide Schlüssel in mindestens einem Byte oder auch nur in der Länge, so wird die Verbindung von der UPIC-Bibliothek sofort wieder abgebaut. Mit diesem Verfahren kann die Echtheit des Schlüssels überprüft werden.

  T-SEL=*transport-selektor*
  Der Transport-Selektor (T-SEL) der Transportadresse adressiert die Partner-Anwendung innerhalb des fernen Systems. Er muss mit den Angaben im fernen System übereinstimmen. Der Transport-Selektor ist ein bis zu 8 Zeichen langer Name. Der angegebene T-SEL überschreibt den beim *Initialize_Conversation* zugewiesenen Wert. Die Angabe von T-SEL ist optional.

  Der Eintrag in der `upicfile` kann mit dem *Set_Partner_Tsel*-Aufruf überschrieben werden.

  T-SEL-FORMAT={T | E | A }
  T-SEL-FORMAT ist der Formatindikator des Transport-Selektors. Gültige Formate sind

  T für TRANSDATA
  E für EBCDIC
  A für ASCII

  T-SEL-FORMAT überschreibt den beim *Initialize_Conversation* zugewiesenen Wert. Die Angabe von T-SEL-FORMAT ist optional.

  Wenn für einen Kommunikationspartner TNS-freier Betrieb festgelegt ist, wird der Wert von TSEL-FORMAT benutzt. Der Eintrag in der `upicfile` kann mit dem *Set_Partner_Tsel_Format*-Aufruf überschrieben werden.

- CONVERTION={IMPLICIT | NO}
  Mit CONVERTION=IMPLICIT geben Sie an, dass beim Senden und Empfangen eine automatische Code-Konvertierung der Benutzerdaten durchgeführt wird. Zur Code-Konvertierung siehe auch Abschnitt „Code-Konvertierung“ auf Seite 68.

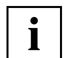
  Geben Sie CONVERTION= nicht an oder verwenden Sie CONVERTION=NO, wird keine automatische Code-Konvertierung durchgeführt.

- Zeilenabschlusszeichen:
  Das Zeichen, das den Eintrag abschließt, ist für die verschiedenen Plattformen, für die die `upicfile` erstellt wird, unterschiedlich:

W W W
  – *Windows-Systeme:*
  Eine Zeile wird durch Carriage Return und Line Feed (Return-Taste) abgeschlossen. Ein Semikolon vor dem Carriage Return-Zeichen ist optional.

- *Unix-Systeme:*
  Die Zeile wird mit einem <newline>-Zeichen (Line Feed) abgeschlossen. Ein Semikolon vor dem <newline>-Zeichen ist optional.
- *BS2000/OSD:*
  Das Zeilenende wird durch ein Semikolon (;) dargestellt. Danach darf kein Leerzeichen mehr folgen.

Falls in einer Zeile (Inhalt des Side Information Eintrags) ein Semikolon steht, reagiert UPIC so, als ob die Zeile dort abgeschlossen wäre und interpretiert den Rest der Zeile als neue Zeile (bis zum nächsten Zeilenabschlusszeichen).



Beachten Sie, dass im BS2000/OSD das nächste Zeilenabschlusszeichen auch wieder ein Semikolon ist. BS2000-Editoren, z.B. EDT haben eine andere Sicht auf Zeilen als UPIC.
Wenn nach dem Semikolon der Zeile *n* im Editor
- noch ein Blank folgt und
- die Zeile *n*+1 mit CD beginnt und mit einem Semikolon endet,

dann sieht UPIC eine Zeile, die mit „ CD“ beginnt und **nicht** mit „CD“.
Der „Symbolic Destination Name“ in dieser Zeile wird nicht gefunden.

Beispiel

Es sollen zwei *Symbolic Destination Names (service1* und *service2*) einer UTM-Cluster-Anwendung konfiguriert werden. Die UTM-Cluster-Anwendung besteht aus drei Knoten-Anwendungen auf den Rechnern CLNODE01, CLNODE02 und CLNODE03. Zusätzlich enthält die upicfile noch einen Eintrag für eine stand-alone UTM-Anwendung UTMAPPL2.

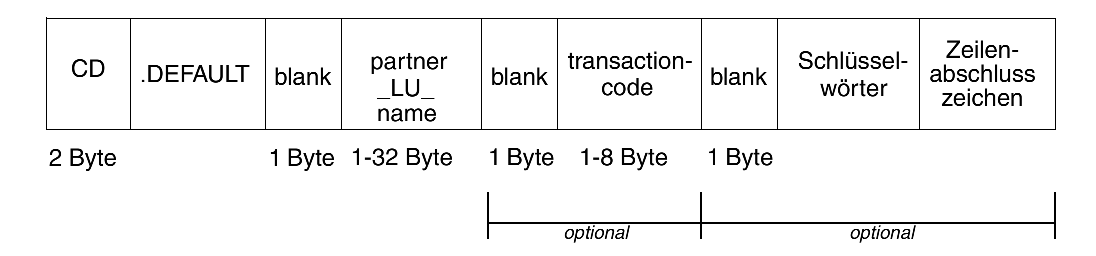
Die Einträge können z.B. so aussehen:

```
* entries for UTM cluster application UTMAPPL1
CDservice1 UTMAPPL1.CLNODE01 TAC1
CDservice1 UTMAPPL1.CLNODE02 TAC1
CDservice1 UTMAPPL1.CLNODE03 TAC1
* entry for stand-alone application UTMAPPL2
SDservice2 UTMAPPL2.D123S234 TAC4
```

Der Transaktioncode TAC1 kann im Programm per *Set_TP_Name* überschrieben werden, so dass sich auch andere TACs ansprechen lassen. Außerdem können wie im Beispiel auch weitere stand-alone UTM-Anwendungen konfiguriert werden (Präfix SD oder HD), diese Einträge müssen aber entweder vor oder nach den oben genannten Einträgen für die UTM-Cluster-Anwendung stehen.

DEFAULT-Server definieren

Sie können für Ihre Client-Anwendung einen DEFAULT-Server bzw. einen DEFAULT-Service definieren (siehe auch Abschnitt „DEFAULT-Server und DEFAULT-Name eines Client“ auf Seite 94). Ein Client-Programm wird mit dem DEFAULT-Server/Service verbunden, wenn im Programm als Symbolic Destination Name ein leerer Name übergeben wird. Im DEFAULT-Eintrag geben Sie statt des Symbolic Destination Name den Wert `.DEFAULT` an. Der DEFAULT-Server-Eintrag muss also folgendes Format haben:

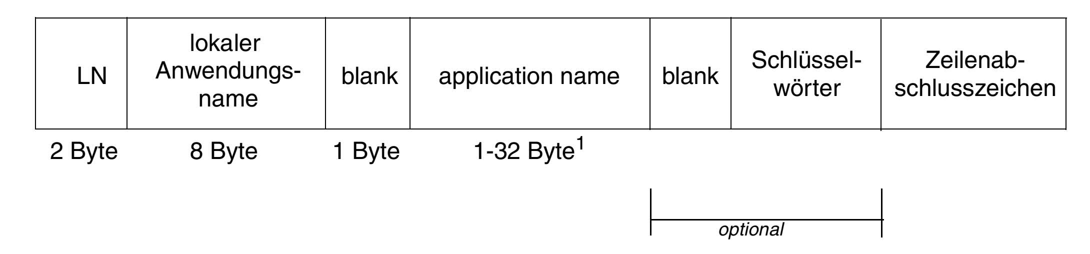


Mit einem solchen Eintrag definieren Sie die UTM-Partner-Anwendung *partner_LU_name* als DEFAULT-Server. Geben Sie einen Transaktionscode an, dann definieren Sie darüber hinaus den zugehörigen Service als DEFAULT-Service. Einen anderen Service am DEFAULT-Server rufen Sie auf, wenn Sie im Programm mit dem Aufruf *Set_TP_Name* einen anderen Transaktionscode setzen (z.B. KDCDISP für den Vorgangs-Wiederanlauf). Die Angabe in *Set_TP_Name* überschreibt den Wert von *transactioncode* im Side Information Eintrag.

6.2.3 Side Information für die lokale Anwendung

Für jede Client-Anwendung können mehrere Einträge in der `upicfile` erstellt werden. Jeder Eintrag definiert einen lokalen Anwendungsnamen, mit dem sich das Client-Programm bei UPIC anmelden kann.

Ein Side Information Eintrag für die lokale Client-Anwendung belegt eine Zeile. Er muss folgendes Format haben:





[1] Bei lokaler Anbindung mit UPIC-Local darf „application name“ nur bis zu 8 Bytes lang sein.

Beschreibung des Eintrags

- Die Zeile beginnt mit dem Kennzeichen LN. LN gibt an, dass es sich um einen Side Information Eintrag für die lokale Client-Anwendung handelt.
- lokaler Anwendungsname
  Hier geben Sie den lokalen Anwendungsnamen an, mit dem sich ein Client-Programm bei UPIC anmeldet. Zwischen dem Kennzeichen LN und dem lokalen Anwendungs-namen darf kein Blank stehen. Der lokale Anwendungsname und der folgende Anwen-dungsname (*application name*) müssen jedoch durch ein Blank getrennt werden.
- application name
  Der application name darf bis zu 32 Zeichen lang sein. Mit dem application name meldet sich die Client-Anwendung beim Transportzugriffssytem an.

  UPIC-Local:
  Der Anwendungsname darf bis zu acht Zeichen lang sein.
- Schlüsselwörter (Angaben optional)
  Mit folgenden Schlüsselwörtern können Sie die UPIC-spezifischen Werte für die lokale Anwendung (siehe hierzu auch „Conversation Characteristics“ auf Seite 51) in der `upicfile` beeinflussen. Mit den Schlüsselwörtern geben Sie die Adressierungsinfor-mationen an.
  Sie können die Schlüsselwörter nach dem *application name* jeweils getrennt durch ein Leerzeichen angeben. Die Reihenfolge und Anzahl der Schlüsselwörter ist beliebig. Mehrere Schlüsselwörter werden mit einem Leerzeichen getrennt.

PORT=*listener-port*

Die Portnummer wird nur für das Adressformat RFC1006 angegeben. Die Portnummer kann einen Wert zwischen 0 bis 32767 annehmen.

Wenn für diesen Kommunikationspartner TNS-freier Betrieb festgelegt ist, wird statt 102 der Wert von PORT als Portnummer benutzt.

Ein Eintrag in der `upicfile` kann mit dem *Specify_Local_Port*-Aufruf überschrieben werden.

X/W UPIC-L Der Wert für PORT wird ignoriert.

T-SEL=*transport-selektor*

Ist der Transport-Selektor (T-SEL) der Transportadresse. Er muss mit den Angaben im fernen System übereinstimmen. Der Transport-Selektor ist ein bis zu 8 Zeichen langer Name. Die Angabe von T-SEL ist optional.

Wenn für einen Kommunikationspartner TNS-freier Betrieb festgelegt ist, wird der Wert von T-SEL benutzt. Der Eintrag in der `upicfile` kann mit dem *Specify_Local_Tsel*-Aufruf überschrieben werden.

X/W UPIC-L Der Wert für T-SEL wird ignoriert.

T-SEL-FORMAT={T | E | A }

T-SEL-FORMAT ist der Formatindikator des Transport-Selektors. Gültige Formate sind

T für TRANSDATA

E für EBCDIC

A für ASCII

Die Angabe von T-SEL-FORMAT ist optional.

Wenn für einen Kommunikationspartner TNS-freier Betrieb festgelegt ist, wird der Wert von TSEL-FORMAT benutzt. Der Eintrag in der `upicfile` kann mit dem *Specify_Local_Tsel_Format*-Aufruf überschrieben werden.

X/W UPIC-L Der Wert für T-SEL-FORMAT wird ignoriert.

- Zeilenabschlusszeichen
  Das Zeilenabschlusszeichen ist plattformabhängig:

W
W
W

  – *Windows-Systeme:*
    Eine Zeile wird durch Carriage Return und Line Feed (Return-Taste) abgeschlossen. Ein Semikolon vor dem Carriage Return-Zeichen ist optional.

X
X
X

  – *Unix-Systeme:*
    Die Zeile wird mit einem <newline>-Zeichen (Line Feed) abgeschlossen. Ein Semikolon vor dem <newline>-Zeichen ist optional.



– *BS2000/OSD:*
  Das Zeilenende wird durch ein Semikolon (;) dargestellt. Danach darf kein Leerzeichen mehr folgen.

Falls in einer Zeile (Inhalt des Side Information Eintrags) ein Semikolon steht, reagiert UPIC so, als ob die Zeile dort abgeschlossen wäre und interpretiert den Rest der Zeile als neue Zeile (bis zum nächsten Zeilenabschlusszeichen).

Wird beim *Enable_UTM_UPIC*-Aufruf für die lokale Anwendung ein lokaler Anwendungsname angegeben, für den es keinen Eintrag in der `upicfile` gibt oder dessen Eintrag ungültig ist, dann übernimmt UPIC den angegebenen Namen als Anwendungsname.

DEFAULT-Name definieren

In der `upicfile` können Sie für Ihre Client-Anwendung einen DEFAULT-Namen definieren (siehe auch Abschnitt „DEFAULT-Server und DEFAULT-Name eines Client“ auf Seite 94). Der DEFAULT-Name wird immer dann verwendet, wenn ein Client-Programm beim Anmelden (*Enable_UTM_UPIC*) einen leeren lokalen Anwendungsnamen übergibt. Im Side Information Eintrag des DEFAULT-Namens geben Sie statt des lokalen Anwendungsnamens den Wert `.DEFAULT` an. Der DEFAULT-Name-Eintrag muss also folgendes Format haben:

LN	.DEFAULT	blank	application name	blank	Schlüssel-wörter	Zeilenab-schlusszeichen
2 Byte		1 Byte	1-32 Byte[1]			
				optional		



[1] Bei lokaler Anbindung mit UPIC-Local darf „application name“ nur bis zu 8 Bytes lang sein.

Immer, wenn ein Client-Programm beim Anmelden einen leeren lokalen Anwendungsnamen an UPIC übergibt, verwendet UPIC diesen Eintrag und meldet das CPI-C-Programm mit dem in *application name* angegebenen Anwendungsnamen beim Transportzugriffssystem an.

Es können sich gleichzeitig mehrere CPI-C-Programme mit dem DEFAULT-Namen bei UPIC anmelden. Diese Programme können sogar mit derselben UTM-Anwendung kommunizieren. Letzteres ist jedoch nur möglich, wenn in der UTM-Anwendung ein LTERM-Pool mit CONNECT-MODE=MULTI für den Anschluss der Client-Anwendung existiert (siehe auch Abschnitt „Mehrfachanmeldungen bei derselben UTM-Anwendung mit demselben Namen“ auf Seite 95).

6.3 Abstimmung mit der Partnerkonfiguration

X/W In Windows- und Unix-Systemen müssen die Angaben im Client-Programm und der Side Information nicht mehr notwendigerweise mit TNS-Einträgen im TNS des lokalen Rechners abgestimmt werden. Wenn Sie UPIC-R ohne CMX (nur mit Socket als Kommunikationssystem) verwenden, ist die Verwendung von TNS-Einträgen weder möglich noch nötig. Wenn Sie UPIC-R mit CMX verwenden, können passende TNS-Einträge erstellt werden, bzw. sind bereits passende TNS-Einträge für den *local_name* und/oder für den *partner_LU_name* in der TNS-Datenbasis vorhanden, so werden an erster Stelle diese TNS-Einträge genommen. Das bedeutet, dass die Aufrufe *Specify_Local_Xxx()* bzw. *Set_Parter_Xxx()* und die Schlüsselwörter der Side Information HOSTNAM.E, IP-ADDRESS, PORT, TSEL und TSEL-FORMAT wirkungslos sind.

B Wenn das Client-Programm im BS2000/OSD abläuft, dann sind ggf. BCMAP-Einträge erforderlich, siehe auch Seite 296.

Zwischen den Angaben im Client-Programm, in der `upicfile` und der UTM-Generierung bestehen Abhängigkeiten. Die folgenden Abschnitte beschreiben, welche Parameter Sie für die Partnerkonfiguration aufeinander abstimmen müssen.

Die nötigen Informationen für das Transportsystem legen Sie entweder direkt in der `upicfile` über Schlüsselwörter oder im Client-Programm durch Funktionsaufrufe fest. Wenn Sie keine dieser Möglichkeiten nutzen, werden voreingestellte Werte verwendet. Die folgende Tabelle gibt einen Überblick über die Voreinstellungen, die Sie in der Side Information oder im Programm ändern können:

Eigenschaft	Funktion	Schlüsselwort	Voreinstellung
lokaler Anwendungsname			
T-SEL	Specify_Local_Tsel	T-SEL=	lokaler Anwendungsname
T-TSEL-Format	Specify_Local_Tsel_Format	T-SEL-FORMAT=	T
Portnummer	Specify_Local_Port	PORT=	102
Transportadresse			
T-SEL	Set_Partner_Tsel	T-SEL=	partnername
T-TSEL-Format	Set_Partner_Tsel_Format	T-SEL-FORMAT=	T
Portnummer	Set_Partner_Port	PORT=	102
Internet-Adresse*	Set_Partner_IP_Address	IP-ADDRESS=	Information aus hosts
Hostname	Set_Partner_Host_Name	HOSTNAME=	prozessorname

Tabelle 14: Eigenschaften Adressierungsinformation

* Die Internet-Adresse hat Vorrang vor dem Hostnamen.

Zwischen den Angaben im Client-Programm oder in der `upicfile` und der Generierung der UTM-Anwendung bestehen folgende Zusammenhänge.

Lokaler Anwendungsname

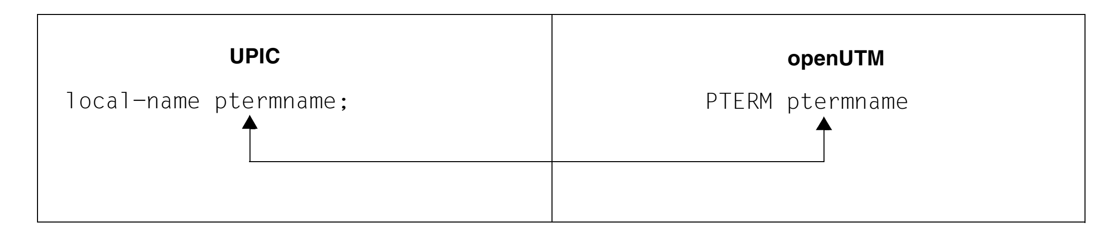
Der lokale Anwendungsname wird bei den Aufrufen *Enable_UTM_UPIC* und *Disable_UTM_UPIC* angegeben. Folgende Fälle sind zu unterscheiden:

– Der lokale Anwendungsname ist in der `upicfile` eingetragen (Kennzeichen LN). Der in diesem Eintrag angegebene Anwendungsname wird direkt an das Transportsystem übergeben.

– Ist der lokale Anwendungsname nicht in der `upicfile` eingetragen, dann wird er von UPIC direkt als Anwendungsname an das Transportsystem übergeben.

Partner auf Unix-, Windows-Systemen oder auf BS2000/OSD ohne BCMAP-Eintrag

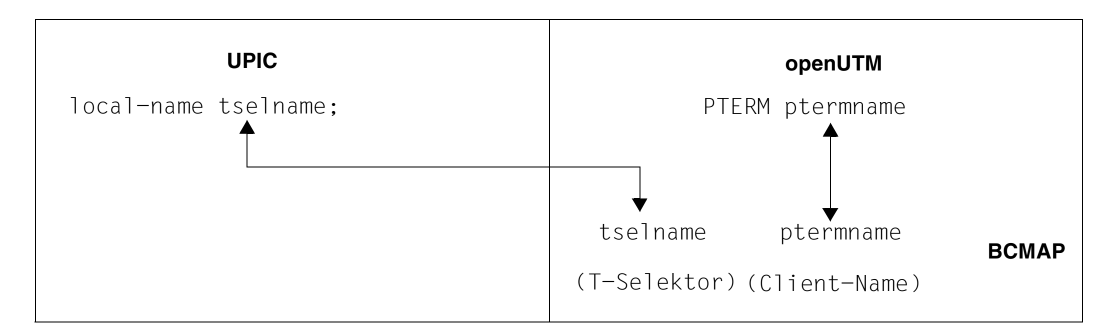
Ist der Partner eine UTM-Anwendung auf einem Unix-System oder einem Windows-System oder eine UTM-Anwendung auf BS2000/OSD, für die keine BCMAP-Einträge erzeugt wurden, dann müssen die Generierungen wie folgt aufeinander abgestimmt sein:



Die beiden PTERM-Namen müssen übereinstimmen. Ist kein PTERM-Name für den Client generiert, dann muss ein LTERM-Pool generiert sein, über den sich der Client anschließen kann.

Partner auf BS2000/OSD mit BCMAP-Eintrag

Ist der Partner eine UTM-Anwendung auf BS2000/OSD, die mit BCMAP-Einträgen arbeitet, müssen die Generierungen wie folgt aufeinander abgestimmt sein:



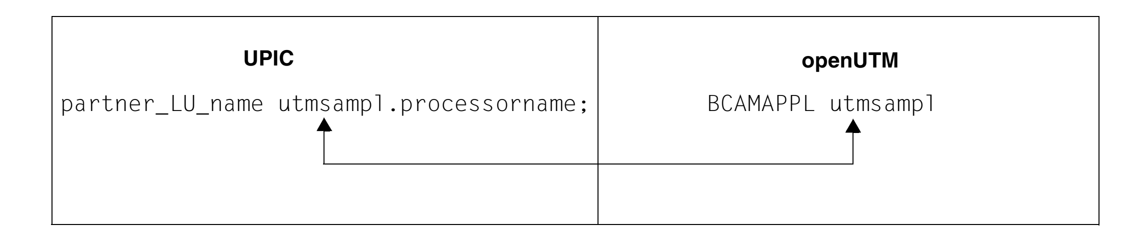
Der T-Selektor der lokalen Anwendung muss mit dem T-Selektor übereinstimmen, der der Client-Anwendung im Server-System zugeordnet ist.

Partner Name

Wenn der *partner_LU_name*, Seite 299) zweistufig angegeben ist (*tselname.processorname*), dann übergibt UPIC diesen Namen direkt an das Transportsystem.

Partner auf Unix-, Windows-Systemen oder auf BS2000/OSD ohne BCMAP-Eintrag

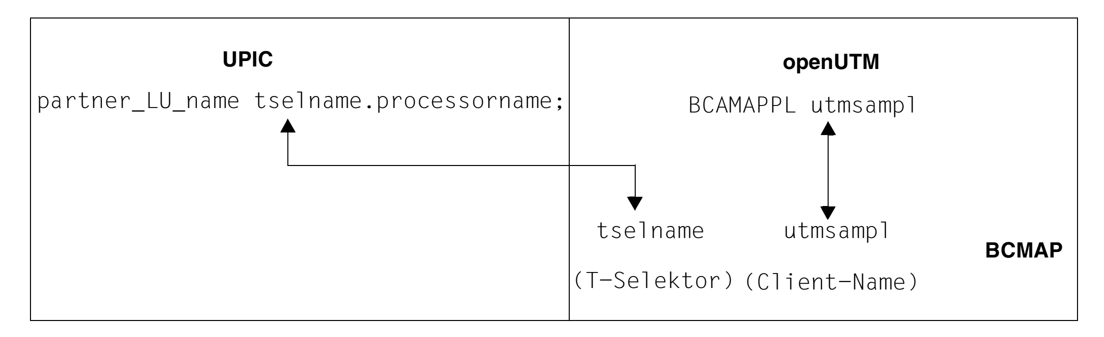
Ist der Partner eine UTM-Anwendung auf einem Unix-System oder einem Windows-System oder eine UTM-Anwendung auf BS2000/OSD, für die keine BCMAP-Einträge erzeugt wurden, dann müssen die Generierungen wie folgt aufeinander abgestimmt sein:



Der *applicationname*, den UPIC an das Transportsystem übergibt, muss dann mit dem BCAMAPPL-Namen der UTM-Anwendung übereinstimmen, über den die Verbindung mit dem Client aufgebaut wird (im Bild *utmsampl*). *processorname* muss im TCP/IP Name Service als Name des fernen Rechners eingetragen sein.

Partner auf BS2000/OSD mit BCMAP-Eintrag

Ist der Partner eine UTM-Anwendung auf BS2000/OSD, die mit BCMAP-Einträgen arbeitet, müssen die Generierungen wie folgt aufeinander abgestimmt sein:



tselname muss mit dem T-Selektor des BCMAP-Eintrags für die UTM-Anwendung am fernen Rechner übereinstimmen.

7 Einsatz von CPI-C-Anwendungen

Dieses Kapitel beschreibt, was Sie vor und während des Einsatzes von CPI-C-Anwendungen beachten müssen, sowie die Maßnahmen, die Sie im Fehlerfall ergreifen können.

7.1 Ablaufumgebung, Binden, Starten

Der Ablauf von CPI-C-Programmen wird durch Umgebungsvariablen bzw. im BS2000/OSD durch Linknamen der Jobvariablen gesteuert. In den folgenden Tabellen sind die für die Steuerung benötigten Variablen aufgeführt:

	Umgebungsvariable	Beschreibung
X/W X/W X/W	UPICPATH	legt das Verzeichnis fest, in dem die Side Information Datei (*upicfile*) abgespeichert ist. Ist die Variable nicht gesetzt, wird die Datei im aktuellen Verzeichnis gesucht.
X/W X/W	UPICFILE	legt den Namen der Side Information Datei fest. Ist die Variable nicht gesetzt, wird der Dateiname *upicfile* gesetzt.
X/W X/W X/W	UPICLOG	legt fest, in welchem Verzeichnis die Logging-Datei abgelegt wird. Der Wert, der angenommen wird, wenn die Variable nicht gesetzt ist, ist plattformabhängig (siehe Abschnitt „UPIC-Logging-Datei“ auf Seite 337).
X/W X/W	UPICTRACE	steuert die Erzeugung eines Trace, siehe Abschnitt „UPIC-Trace“ auf Seite 338.

Linknamen der Jobvariablen	**Beschreibung**
UPICPAT	legt den teilqualifizierten Dateinamen [:catid:$*userid*.<*Teilnamen*>] fest, unter dem die Side Information Datei (*upicfile*) abgespeichert ist. Ist die Variable nicht gesetzt, wird die Datei unter $*userid* gesucht.
UPICFIL	legt den rechten Teil des Namen der Side Information Datei fest. Ist die Variable nicht gesetzt, wird der Dateiname upicfile gesetzt. Der vollständige Dateiname setzt sich zusammen aus UPICPAT.UPICFIL. Sind weder UPICPAT noch UPICFIL gesetzt, so lautet er „$*userid*.UPICFILE“.
UPICLOG	legt fest, unter welchem teilqualifizierten Dateinamen die Logging-Datei abgelegt wird. Der Wert, der angenommen wird, wenn die Variable nicht gesetzt ist, ist plattformabhängig (siehe Abschnitt „UPIC-Logging-Datei“).
UPICTRA	steuert die Erzeugung eines Trace, siehe Abschnitt „UPIC-Trace“ auf Seite 338.

In den folgenden Abschnitten ist plattformabhängig beschrieben, was Sie beim Erzeugen und beim Einsatz einer CPI-C-Anwendung an Ihrem System beachten müssen.

7.1.1 Einsatz in Windows-Systemen

Bei der Erstellung und beim Einsatz von CPI-C-Anwendungen müssen Sie die in den Abschnitten „Übersetzen, Binden, Starten“ auf Seite 321 und „Ablaufumgebung, Umgebungsvariablen“ auf Seite 322 beschriebenen Besonderheiten beachten.
Beim Erstellen und beim Einsatz von UPIC-Local-Anwendungen auf Windows-Systemen sind weitere Spezifika zu berücksichtigen. Sie sind in Abschnitt „Besonderheiten beim Einsatz von UPIC-Local auf Windows-Systemen“ auf Seite 324 beschrieben.

7.1.1.1 Übersetzen, Binden, Starten

Beim Übersetzen und Binden von CPI-C-Anwendungen auf Windows-Systemen müssen Sie folgendes berücksichtigen:

- Jedes CPI-C-Programm benötigt zum Übersetzen folgende Include-Dateien:

  ```
  #include <WINDOWS.H>
  #include <upic.h>
  ```

  Die Include-Datei `upic.h` befindet sich im Verzeichnis *upic-dir*`\include`. *upic-dir* ist das Installationsverzeichnis von openUTM-Client mit Trägersystem UPIC.

  Die oben angegebene Reihenfolge der Includes muss eingehalten werden. Es wird empfohlen, das Programm mit der Option `__STDC__ (ANSI)` zu übersetzen.

- Beim Compilieren von CPI-C-Programmen (nur UPIC-Remote) müssen Sie folgende Compileroptionen unbedingt setzen: `UTM_ON_WIN32`

  Die Wirkung dieser Option können Sie der Include-Datei `upic.h` entnehmen. Sie befindet sich im Verzeichnis *upic-dir*`\include`. *upic-dir* ist das Installationsverzeichnis von openUTM-Client mit Trägersystem UPIC.

- Ein CPI-C-Programm besteht aus einer Reihe von Modulen, die als ein Programm gebunden werden müssen. Folgende Objekte sind zum Binden notwendig:

  – main-Programm des Anwenders
  – Anwendermodule
  – Für Programme, die PCMX verwenden wollen:
  die Bibliothek `upicw32.lib`, die sich im Verzeichnis *upic-dir*`\SYS` befindet.
  – Für Programme, die die Socket-Schnittstelle verwenden wollen:
  die Bibliothek `upicws32.lib` (socket), die sich im Verzeichnis *upic-dir*`\SYS` befindet.

- Nachdem die Ablaufumgebung (siehe nächster Abschnitt) bereitgestellt wurde, starten Sie ein CPI-C-Programm wie jedes andere Programm in Windows-Systemen.

7.1.1.2 Ablaufumgebung, Umgebungsvariablen

Zur Steuerung von CPI-C-Anwendungen dienen die Umgebungsvariablen, die in der Tabelle auf Seite 319 aufgeführt sind.

In der Variablen UPICTRACE kann der Pfadname mit Leerzeichen angegeben werden. Wenn Leerzeichen verwendet werden, muss der Pfadname in doppelte Hochkommata eingeschlossen werden. Sind keine Leerzeichen im Pfadnamen, können doppelte Hochkommata auch verwendet werden.

Unter Windows-Systemen setzen Sie Umgebungsvariablen wie folgt:

– Geben Sie nacheinander `Start/Einstellungen/Systemsteuerung/System` ein, wählen Sie im Dialogfeld `Systemeigenschaften` die Registerkarte `Erweitert` und klicken Sie dort auf `Umgebungsvariable`.

– Erzeugen/Erweitern Sie im Dialgogfeld `Umgebungsvariable` die Umgebungsvariablen. Diese Einstellungen bleiben unter Windows-Systemen bis zur nächsten Änderung gültig.

Bei anderen Windows-Systemen gehen Sie analog vor, die Menübefehle und -bezeichnungen können sich dabei etwas unterscheiden.

Es gibt Benutzervariablen, die nur für die aktuelle Benutzerkennung gelten, und Systemvariablen, die für alle Benutzer gelten. Wollen Sie eine UPIC-Anwendung als Service betreiben (ein Service läuft ohne Benutzerumgebung), so müssen Sie Systemvariablen setzen.

Alle diese Umgebungsvariablen können auch mit Hilfe der Datei `UPIC.INI` gesetzt werden.

UPIC.INI

Die Umgebungsvariablen, über die eine CPI-C-Anwendung gesteuert wird, können mit Hilfe der Datei `UPIC.INI` gesetzt werden. Wird die Datei `UPIC.INI` verwendet, dann muss sie im Windows-Verzeichnis stehen und folgenden Aufbau haben:

```
[UPICW32DLL]
UPICPATH=verzeichnis
UPICTRACE=schalter
UPICLOG=verzeichnis
UPICFILE=name-side-information-datei
```

Beispiel

```
[UPICW32DLL]
UPICPATH=C:\UPIC
UPICTRACE=-SX -dC:
UPICLOG=C:\UPIC\TMP
UPICFILE=upicfile
```

Die Einträge in `UPIC.INI` werden nur ausgewertet, wenn die betreffende Umgebungsvariable nicht gesetzt wurde.

Die Verwendung der Datei `UPIC.INI` wird nicht mehr empfohlen. Da sie im Windows-Verzeichnis stehen muss, haben nicht alle Benutzer uneingeschränkten Zugriff auf die Datei. Benutzer, die der Gruppe `User` angehören, haben z.B. kein Schreibrecht auf die Datei.

Registry Einträge

Die UPIC-Bibliothek beherrscht den Mechanismus des „IniFileMapping“. Nähere Informationen dazu finden Sie z.B. in der „MSDN Library Visual Studio 6.0 - Platform SDK - Windows Base Services“ unter „WritePriviteProfileString()“. Der zugehörige Key lautet `HKCU\Software\FSC\UPIC\UPICW32DLL`.

Sie können unter dem Subkey UPICW32DLL (entspricht dem Eintrag „Section“ in UPIC.INI) die Werte UPICPATH, UPICTRACE, UPICLOG und UPICFILE einrichten und im Datenfeld die Werte eingeben, wie oben beschrieben.

Die Registry-Werte werden nur ausgewertet, wenn die betreffende Umgebungsvariable nicht gesetzt wurde. Sie werden aber vor den Einträgen in der UPIC.INI ausgewertet.

Betriebsmittel eines CPI-C-Programms

- Für die Trace-Datei wird ein File-Deskriptor ständig belegt.
- Wird in die Logging-Datei geschrieben, dann wird nur während des Schreibens ein File-Deskriptor belegt.
- Zum Lesen aus der `upicfile` wird nur während des Aufrufs *Enable_UTM_UPIC* ein File-Deskriptor benötigt.
- Hinzu kommen die Betriebsmittel, die vom Transportsystem belegt werden.

7.1.1.3 Besonderheiten beim Einsatz von UPIC-Local auf Windows-Systemen

Beim Einsatz von UPIC-Local-Anwendungen auf Windows-Systemen sind die im Folgenden beschriebenen Besonderheiten zu beachten.

UPIC-Local-Anwendungen binden

Zum Binden von UPIC-Local-Anwendungen auf Windows-Systemen werden die folgenden Bibliotheken ausgeliefert:

- *utmpath*`\upicl\sys\libupicl.lib`, die zu jedem Client-Programm gebunden werden muss (statt `upicw32.lib`) und
- *utmpath*`\xatmi\sys\libxtclt.lib`, die zusätzlich zu XATMI-Programmen gebunden werden muss.

Nähere Informationen zu *utmpath* entnehmen Sie dem openUTM-Handbuch „Einsatz von openUTM-Anwendungen unter Unix- und Windows-Systemen“.

Ablaufumgebung

Für den Ablauf der UPIC-Local-Clients werden die dynamischen Bibliotheken *utmpfad*`\ex\libupicl.dll` und *utmpfad*`\ex\libxtclt.dll` benötigt.

Diese DLLs werden über die Umgebungsvariable PATH gefunden. Die Umgebungsvariable PATH muss nach der Installation von openUTM manuell entsprechend erweitert werden.

UPIC-Local-Client mit Visual C++ konfigurieren

Im folgenden wird kurz dargestellt, wie Sie mit dem Visual C++ Developer Studio ein UPIC-Local-Client-Projekt konfigurieren können. UPIC-Local ist Bestandteil von openUTM für Windows-Systeme und setzt mindestens die Version 5 des Visual C++ Developer Studios voraus. Bei anderen Developer Studio-Versionen gehen Sie analog vor, die Menübefehle und -bezeichnungen können sich dabei etwas unterscheiden.

> **i** Client-Projekte, die mit dem openUTM Quick Start Kit ausgeliefert werden, sind wie hier beschrieben konfiguriert.

Zur Konfigurierung des Projektes wählen Sie im Menü *Projekt* des Visual C++ Developer Studios den Befehl *Einstellungen* aus. Am Bildschirm wird das Dialogfeld *Projekteinstellungen* angezeigt. Jetzt gehen Sie wie folgt vor:

1. UPIC-Local-Bibliotheken `libupicl.lib` und `libxtclt.lib` einbinden:

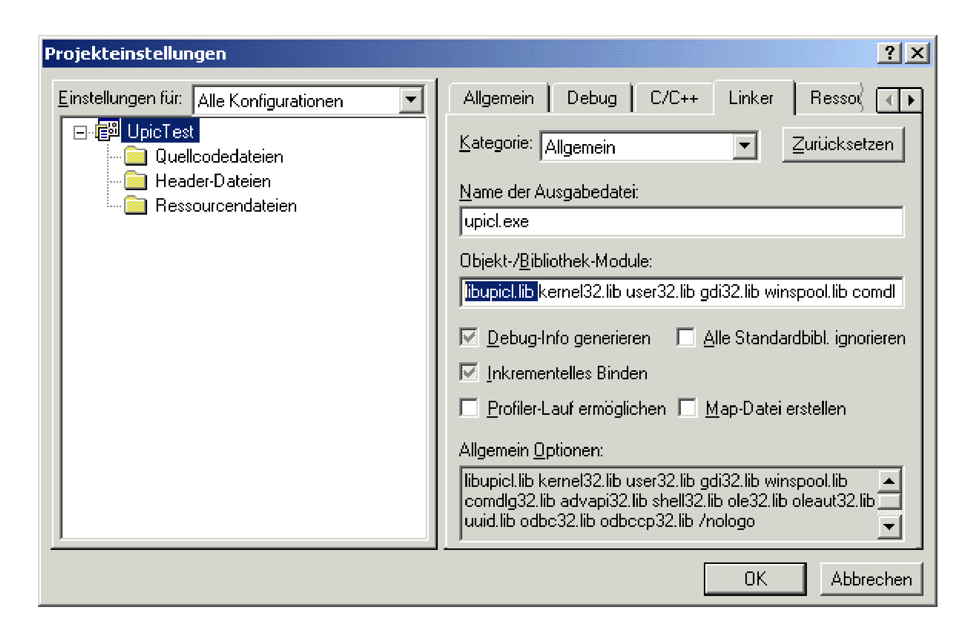
   Wählen Sie das Registerblatt *Linker* aus und stellen Sie sicher, dass in der Liste *Einstellungen für* der Punkt *Alle Konfigurationen* markiert ist.

   In der Liste *Kategorie* stellen Sie die Kategorie *Allgemein* ein, tragen bei Name der Ausgabedatei den gewünschten Namen ein (hier `upicl.exe`) und erweitern die Angaben im Eingabefeld *Objekt-/Bibliothek-Module* um folgende Bibliotheken:

   – `libupicl.lib` bei der Konfigurierung von CPI-C-Clients

   – `libxtclt.lib` und `libupicl.lib` bei der Konfigurierung von XATMI-Clients (Reihenfolge beachten, `libxtclt.lib` muss vor `libupicl.lib` stehen). Als Trennzeichen ist jeweils ein Leerzeichen einzugeben.

   Diese Bibliotheken müssen vor allen schon vorhandenen `*.lib`-Dateien eingefügt werden. Damit Sie nicht den kompletten Pfadnamen eintippen müssen, geben Sie im Developer Studio in *Extras/Optionen* die Suchpfade ein.

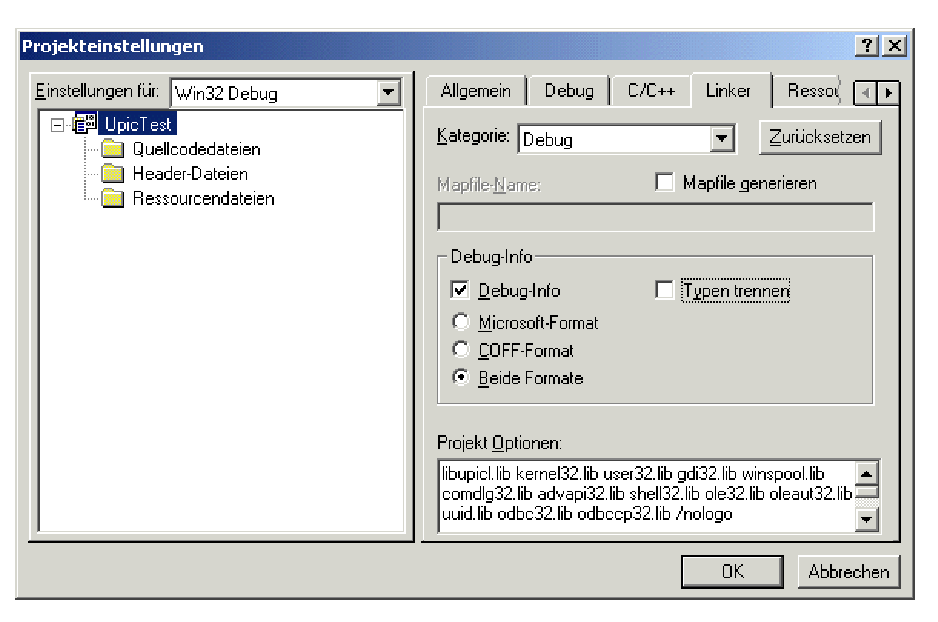


W 2. Debugger Information konfigurieren:

W Wählen Sie das Registerblatt *Linker* aus und markieren jetzt in der Liste *Einstellungen*
W *für* den Punkt *Win32Debug*.

W In der Liste *Kategorie* stellen Sie die Kategorie *Debug* ein und aktivieren Sie in *Debug Info*
W die Optionen *Debug Info* und *Beide Formate*.

W



W 3. Bestätigen Sie Ihre Angaben, indem Sie auf die Schaltfläche **OK** klicken.

7.1.2 Einsatz in Unix-Systemen

Bei der Erstellung und beim Einsatz von CPI-C-Anwendungen müssen Sie die in den Abschnitten „Übersetzen, Binden, Starten“ auf Seite 327 und „Ablaufumgebung, Umgebungsvariablen“ auf Seite 328 beschriebenen Besonderheiten beachten.
Beim Erstellen und beim Einsatz von UPIC-Local-Anwendungen auf Unix-Systemen sind weitere Spezifika zu berücksichtigen. Sie sind in Abschnitt „Besonderheiten beim Einsatz von UPIC-Local auf Unix-Systemen“ auf Seite 329 beschrieben.

7.1.2.1 Übersetzen, Binden, Starten

Beim Übersetzen und Binden von CPI-C-Anwendungen auf Unix-Systemen müssen Sie folgendes berücksichtigen:

- Jedes CPI-C-Programm benötigt zum Übersetzen folgende Include-Datei:

  ```
  #include <upic.h>
  ```

  Die Include-Datei befindet sich im Unterverzeichnis `include` des UPIC-Installationsverzeichnisses.

- Ein CPI-C-Programm besteht aus einer Reihe von Modulen, die mit dem C-Compiler Ihres Systems zu einem Programm gebunden werden. Folgende Objekte sind zum Binden notwendig:
  - main-Programm des Anwenders
  - Anwendermodule

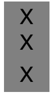
  Für Programme, die PCMX verwenden:
  - Die Systembibliotheken `nsl.so`, `dl.so`, `socket.so` (nicht auf jedem System) und `cmx.so`. Die Bibliothek `cmx.so` muss auf jeden Fall vor der Bibliothek `nls.so` eingebunden werden.
  - Die Bibliothek `libupiccmx.a` (statisch) oder die Bibliothek `libupiccmx.so` (dynamisch), die sich im Verzeichnis *upic-dir*/`sys` befindet.

  Für Programme, die PCMX nicht verwenden:
  - Die Systembibliotheken `nsl.so` und `dl.so`. Auf wenigen Systemen auch `socket.so`.
  - Die Bibliothek `libupicsoc.a` (statisch) oder die Bibliothek `libupicsoc.so` (dynamisch), die sich im Verzeichnis *upic-dir*/`sys` befinden.

  Für Programme, die PCMX nicht verwenden und Threading wollen:
  - Die Systembibliotheken `nsl.so`, `dl.so` und `socket.so`
  - Die Bibliothek `libupicsocmt.a` (statisch) oder die Bibliothek `libupicsocmt.so` (dynamisch), die sich im Verzeichnis *upic-dir*/`sys` befinden.

  Ein Beispiel für alle benötigten Bibliotheken und Bindeoptionen finden Sie im Makefile für das Beispielprogramm `uptac.c` im Verzeichnis *upic-dir*/`sample`.

  i Bei einigen HP-Systemen besitzen die Bibliotheken die Endung `.sl` anstatt `.so`.



- Ein CPI-C-Programm starten Sie wie jedes andere Programm in Unix-Systemen durch Eingabe des Programmnamens (beachten Sie, dass die UTM-Anwendung vorher gestartet sein muss).

7.1.2.2 Ablaufumgebung, Umgebungsvariablen

Zur Steuerung von CPI-C-Anwendungen dienen die Umgebungsvariablen, die in der Tabelle „Umgebungsvariable“ auf Seite 319 aufgeführt sind.

Die Umgebungsvariablen können Sie wie folgt setzen:

```
UPICPATH=verzeichnis
UPICTRACE=schalter
UPICLOG=verzeichnis
UPICFILE=name-side-information-datei
export UPICPATH UPICTRACE UPICLOG UPICFILE
```

Betriebsmittel eines CPI-C-Programms

- Für die Trace-Datei wird ein File-Deskriptor ständig belegt.
- Wird in die Logging-Datei geschrieben, dann wird nur während des Schreibens ein File-Deskriptor belegt.
- Zum Lesen aus der `upicfile` wird nur während des Aufrufs *Enable_UTM_UPIC* ein File-Deskriptor benötigt.
- Hinzu kommen die Betriebsmittel, die vom Transportsystem belegt werden.

Signale

Signalbehandlungsroutinen dürfen Sie in einem CPI-C-Programm nur für die Signale SIGHUP, SIGINT und SIGQUIT schreiben. Die CPI-C-Bibliotheksfunktionen werden durch diese drei Signale nicht unterbrochen. Diese Signalbehandlung wird erst nach dem Ende der aktuellen CPI-C-Funktion wirksam.

Alle anderen Signale sind verboten!

7.1.2.3 Besonderheiten beim Einsatz von UPIC-Local auf Unix-Systemen

X
X

Beim Einsatz von UPIC-Local-Anwendungen auf Unix-Systemen sind zusätzlich die im Folgenden beschriebenen Besonderheiten zu beachten.

X **Binden von UPIC-Local-Anwendungen in Unix-Systemen**

X
X
X

Bei der lokalen Anbindung einer CPI-C-Client-Anwendung an eine UTM-Anwendung auf einem Unix-System müssen Sie statt der Bibliotheken `libupiccmx.a` bzw. `libupiccmx.so` die Bibliothek `libupicipc.a` (statisch) bzw. `libupicipc.so` (dynamisch) einbinden.

X Auf Linux-Systemen muss zusätzlich die Option *–lcrypt* angegeben werden.

X **Umgebungsvariablen**

X
X
X

Für die Steuerung einer UPIC-Local-Anwendung wird auch die Umgebungsvariable UTMPATH ausgewertet. UTMPATH muss den Namen des Verzeichnisses enthalten, in dem openUTM installiert ist.

X **Betriebsmittel**

X
X
X

Bei lokaler Anbindung wird zur Kommunikation mit der UTM-Anwendung Shared Memory verwendet. Der Zugriff erfolgt über „shared memory keys“ und wird mit Hilfe eines Semaphors serialisiert. Für Shared Memory wird ein zusätzlicher File-Deskriptor belegt.

7.1.3 Einsatz unter BS2000/OSD

Beim Einsatz von CPI-C-Anwendungen im BS2000/OSD beachten Sie bitte die nachfolgend aufgeführten Besonderheiten.

Übersetzen, Binden, Starten

Beim Übersetzen und Binden von CPI-C-Anwendungen auf BS2000/OSD-Systemen gilt Folgendes:

- Jedes CPI-C-Programm benötigt zum Übersetzen folgende Include-Datei:

  ```
  #include <UPIC.H>
  ```

  Die Include-Datei befindet sich in der Bibliothek `$`*userid*`.SYSLIB.UTM-CLIENT.061`.

  userid ist die Kennung, unter der openUTM-Client installiert wurde.

- Ein CPI-C-Programm besteht aus einer Reihe von Modulen, die als ein Programm gebunden werden müssen. Folgende Objekte sind zum Binden notwendig:
  - main-Programm des Anwenders
  - Anwendermodule
  - Für Programme, die CMX verwenden wollen:
    - Die Systembibliotheken *$sysid*`.SYSLNK.CRTE` und `$sysid.SYSLIB.CMX.013`
    - Die Bibliotheken *$userid*`.SYSLIB.UTM-CLIENT.061.WCMX` und `$`*userid*`.SYSLIB.UTM-CLIENT.061`
  - Für Programme, die Sockets verwenden wollen:
    - Die Systembibliothek *$sysid*`.SYSLNK.CRTE`
    - Die Bibliotheken *$userid*`.SYSLIB.UTM-CLIENT.061`
    - Die entsprechenden Socket-Bibliotheken

- Ein CPI-C-Programm starten Sie wie jedes andere Programm in BS2000/OSD mit dem Kommando `START-EXECUTABLE-PROGRAM`.

B **Ablaufumgebung**

B Der Ablauf von CPI-C-Anwendungen unter BS2000/OSD wird über die Jobvariablen
B gesteuert. Die Linknamen der Jobvariablen sind in der Tabelle auf Seite 320 aufgeführt.
B Diese können Sie z.B. wie folgt setzen:

```
B /SET-JV-LINK LINK-NAME=*UPICPAT,JV-NAME=UPICPATH
B /MODIFY-JV JV[-CONTENTS]=UPICPATH,SET-VALUE='prefix'
B /SET-JV-LINK LINK-NAME=*UPICFIL,JV-NAME=UPICFILE
B /MODIFY-JV JV[-CONTENTS]=UPICFILE,SET-VALUE='filename'
B /SET-JV-LINK LINK-NAME=*UPICLOG,JV-NAME=UPICLOG
B /MODIFY-JV JV[-CONTENTS]=UPICLOG,SET-VALUE='prefix'
B /SET-JV-LINK LINK-NAME=*UPICTRA,JV-NAME=UPICTRACE
B /MODIFY-JV JV[-CONTENTS]=UPICTRACE,SET-VALUE='schalter'
```

B *Beispiel*

```
B /SET-JV-LINK LINK-NAME=*UPICTRA,JV-NAME=UPICTRACE
B /MODIFY-JV JV[-CONTENTS]=UPICTRACE,SET-VALUE='-r 128'
```

B Beachten Sie, daß die mit SET-JV-LINK hergestellte Zuweisung des Kettungsnamens nach
B dem LOGOFF verlorengeht. Set-VALUE='-r 128' steuert den Trace (siehe Abschnitt „UPIC-
B Trace“ auf Seite 338).

7.2 Behandlung von CPI-C-Partnern durch openUTM

Bei einer Anbindung an eine UTM-Anwendung über CPI-C können einige Funktionen von openUTM nicht oder nur anders genutzt werden.

Folgende Funktionen sind betroffen:

– INPUT-Exit und Event-Service BADTAC

  Bei Eingaben von einem CPI-C-Client ruft openUTM den INPUT-Exit und BADTAC nicht auf.

– FPUT

  Es ist nicht möglich eine asynchrone Nachricht mittels FPUT an einen CPI-C-Client zu senden. Der KDCS-Aufruf liefert 44Z als Returncode.

– PEND RS

  Für einen CPI-C-Client wird PEND RS unter Umständen wie PEND FR behandelt, Näheres siehe auch im openUTM-Handbuch „Anwendungen programmieren mit KDCS“.

7.3 Verhalten im Fehlerfall

In diesem Abschnitt ist beschrieben, wie sich die Beendigung einer UTM-Server-Anwendung bzw. einer CPI-C-Client-Anwendung auf den Kommunikationspartner auswirkt. Außerdem wird erklärt, was Sie tun müssen, um nach einer Fehlersituation wieder einen Grundzustand für eine erfolgreiche Programm-Programm-Kommunikation herzustellen.

Eine UTM-Anwendung beendet sich

Falls sich die UTM-Anwendung beendet, merkt dies das CPI-C-Client-Programm beim nächsten Aufruf an der Kommunikationsschnittstelle. Dabei können folgende Fälle unterschieden werden:

– Bei einem *Receive*-Aufruf wird ein Verbindungsabbau erkannt oder
– bei einem Aufruf an der Kommunikationsschnittstelle wird erkannt, dass sich die Anwendung beendet hat, wodurch sich automatisch auch die Conversation beendet hat.

In beiden Fällen wird als Ergebnis CM_DEALLOCATED_ABEND zurückgeliefert.

Ein CPI-C-Programm beendet sich abnormal

Die UTM-Anwendung bekommt die Programmbeendigung in der Regel durch einen Verbindungsabbau angezeigt. In diesem Fall sind keine zusätzlichen Aktivitäten erforderlich.

Falls der UTM-Anwendung kein Verbindungsabbau angezeigt wird, bleibt die Verbindung aus Sicht von openUTM bestehen. Es sind zwei Fälle zu unterscheiden:

– Auf der UTM-Seite ist für die Client-Anwendung ein PTERM oder ein LTERM-Pool mit TPOOL ...,CONNECT-MODE=SINGLE generiert. In diesem Fall kann openUTM die angeschlossenen Clients unterscheiden. Sobald ein Client (nach einem Verbindungsverlust) mit demselben Namen wieder eine Verbindung aufbauen will, baut openUTM die alte Verbindung ab und weist den Verbindungsaufbauwunsch zurück. Ein darauf folgender erneuter Verbindungsaufbauwunsch des Clients wird dann wieder akzeptiert.

– Auf der UTM-Seite ist für die Client-Anwendung ein LTERM-Pool mit TPOOL ..., CONNECT-MODE=MULTI generiert. In diesem Fall können sich von einem Rechner aus mehrere Clients mit demselben Namen bei der UTM-Anwendung anmelden. Die UTM-Anwendung kann dann nicht mehr erkennen, ob sich ein Client neu oder nach einem Verbindungsverlust anmelden will. Eine verlorengegangene Verbindung, für die der UTM-Anwendung kein Verbindungsabbau angezeigt wurde, muss in diesem Fall explizit durch die Administration abgebaut werden. D.h. openUTM baut die „verlorengegangene“ Verbindung beim folgenden Versuch des Client, eine Verbindung aufzubauen, nicht selbst ab.

Upic-Local

Folgender Fall kann auftreten:

Die UTM-Anwendung hat nichts von der Beendigung des CPI-C-Prozesses gemerkt. Sobald sich das CPI-C-Programm wieder mit demselben Programmnamen an openUTM anmeldet, baut openUTM die alte Verbindung ab und akzeptiert die neue Verbindung.

Schwerwiegender Fehler im CPI-C-Programm

Tritt während des Ablaufs des CPI-C-Programms ein schwerwiegender Fehler auf, der eine sinnvolle Fortsetzung nicht ermöglicht, wird der Prozess abnormal beendet (in Windows-Systemen mit `FatalAppExit`; in Unix-Systemen mit *abort*). Außerdem wird folgende Fehlermeldung in die UPIC-Logging-Datei geschrieben:

```
UPIC: internal error <reason>
```

Die Fehlermeldungen, die auf der CPI-C-Seite auftreten können, sind in der folgenden Tabelle beschrieben.

<reason>	**Bedeutung**
1	Beim Senden von Restdaten ist der Wert für die Datenlänge negativ
9	Das Signal SIGTRAP ist aufgetreten
10	Fehler beim Verbindungsaufbau
11	Fehler beim Empfangen der Bestätigung für den Verbindungsaufbau
12	Nachricht ungleich Verbindungsaufbau erhalten
13	Fehler beim Senden von Daten
14	Fehler beim Empfangen von Daten
15	Empfangen einer ungültigen Nachricht
16	Fehler beim Verbindungsabbau

Zur Fehlerdiagnose siehe auch Abschnitt „Diagnose“ auf Seite 337.

UPIC-Local



Bei der lokalen Kommunikation über UPIC-Local können darüber hinaus Fehlermeldungen auftreten, die mit den Buchstaben „IPC“ beginnen. Diese sind durch openUTM verursacht. Sie sind im openUTM-Handbuch „Meldungen, Test und Diagnose in UNIX- und Windows-Systemen“ bei den Dump-Fehlercodes beschrieben.

Zur Fehlerdiagnose ist der Dump (z.B. core-Dump in Unix-Systemen) zusammen mit dem gebundenen Programm sowie der Inhalt der UPIC-Trace-Datei und der UPIC-Logging-Datei notwendig.

Nachrichtenaustausch bei programmiertem PEND ER/FR

Wenn im UTM-Teilprogrammlauf ein programmierter PEND ER/FR durchgeführt wurde, können die vor dem PEND ER/FR mit MPUT gesendeten Teilnachrichten empfangen werden. Dies geschieht mit dem Aufruf *Receive* bzw. *Receive_Mapped_Data* (solange bis das Ergebnis CM_DEALLOCATED_ABEND ist).

Nachrichtenaustausch bei SYSTEM PEND ER

Falls der UTM-Vorgang im Fehlerfall auf PEND ER läuft, wird beim Aufruf *Receive* bzw. *Receive_Mapped_Data* das Ergebnis CM_DEALLOCATED_ABEND geliefert. Zusätzlich wird eine Fehlermeldung in die Logging-Datei geschrieben (siehe auch Abschnitt „UPIC-Logging-Datei“ auf Seite 337).

Mit dem Aufruf *MPUT ES* (error system) kann in einem Dialog-Teilprogramm eine eigene Fehlermeldung für einen UPIC-Client erzeugt werden (siehe auch openUTM-Handbuch „Anwendungen programmieren mit KDCS“, Aufruf *MPUT ES*), die der UPIC-Client mit dem Aufruf *Receive* bzw. *Receive_Mapped_Data* lesen kann. In diesem Fall wird keine Fehlermeldung in die Logging-Datei geschrieben.

Probleme beim Verbindungsaufbau

Probleme beim Verbindungsaufbau zur UTM-Partner-Anwendung sind daran zu erkennen, dass der Aufruf *Allocate* nicht mit dem Ergebnis CM_OK endet. In einem solchen Fall sollten Sie folgendes überprüfen:

- Überprüfen Sie mit einem `ping`-Kommando, ob überhaupt eine Netzverbindung zwischen Client und Server zustande kommen kann.

  X/W Sie rufen das Kommando `ping` auf mit:

  X/W ping <internetadresse> oder ping <hostname>

  X/W `ping` muss in Ihrem Pfad liegen, d.h. die Variable PATH muss entsprechend gesetzt sein.

  B Unter BS2000/OSD rufen Sie ping wie folgt auf:

  ```
  /START-EXECUTABLE-PROGRAM                                   -
         FROM-FILE=*LIBRARY-ELEMENT                           -
         (LIBRARY=$.SYSPRG.BCAM.XXX,ELEMENT-OR-SYMBOL=PING)
  ```

- Überprüfen Sie das TCP/IP Protokoll. Dazu können Sie eine der Standard-Anwendungen `telnet` oder `ftp` benutzen.

  X/W Diese Kommandos rufen Sie auf mit:

  X/W `telnet` *internetadresse* oder `telnet` *hostname*
  X/W `ftp` *internetadresse* oder `ftp` *hostname*

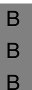


Die Anwendungen müssen in Ihrem Pfad liegen, d.h. die Variable PATH muss entsprechend gesetzt sein.

Unter BS2000/OSD werden die Anwendungen aufgerufen mit:

START-TELNET
START-FTP

- Überprüfen Sie, ob in der UTM-Partner-Anwendung die erforderlichen Betriebsmittel zur Verfügung stehen. Z.B. darf der LTERM-Pool bzw. der LTERM-Partner, über den sich der Client anschließen will, nicht gesperrt sein. Siehe dazu auch das openUTM-Handbuch „Anwendungen generieren“.
- Überprüfen Sie, ob im lokalen System die erforderlichen Betriebsmittel zur Verfügung stehen.In jedem Fall sollten Sie auch die lokale Generierung (Side Information und ggf. TNS) sowie die Generierung des Partners (openUTM und ggf. TNS) überprüfen.



Bei einer Konfiguration, die BCMAP-Einträge im BS2000/OSD erfordert, müssen Sie beachten, dass das Kommando BCMAP keine Update-Funktion besitzt, d.h. dass BCMAP-Einträge zuerst gelöscht und dann neu eingetragen werden müssen. Näheres zum Kommando BCMAP finden Sie in den BCAM-Handbüchern.

7.4 Diagnose

Folgende Unterlagen werden für die Diagnose benötigt:

- eine genaue Beschreibung der Fehlersituation
- Angabe, welche Software mit welchen Versionsständen eingesetzt wurde
- genaue Angabe des Rechnertyps
- das CPI-C-Programm als Source
- die Side Information Datei (`upicfile`)
- die UPIC-Logging-Datei und die UPIC-Trace-Dateien, siehe folgende Abschnitte
- die CMX- bzw. PCMX-Trace-Dateien
- X bei Unix-Systemen die core-Dateien mit zugehörigen Phasen

Bei Fehlern, die in Zusammenhang mit der UTM-Partner-Anwendung stehen, werden zusätzliche openUTM-Unterlagen benötigt:

- KDCDEF-Generierung und UTM-Diagnosedump der UTM-Partner-Anwendung
- Mitschnitte der Ausgaben auf die Standardausgabe und die Standardfehlerausgabe
- X/W Unix- und Windows-Systeme: *stderr*, *stdout*,
- B BS2000: SYSLST, SYSLOG, SYSOUT.

7.4.1 UPIC-Logging-Datei

Zur Erleichterung der Diagnose führt das Trägersystem UPIC eine Logging-Datei. In diese Datei wird z.B. eine UTM-Fehlermeldung geschrieben, falls die UTM-Anwendung eine Conversation abnormal beendet. Die Logging-Datei wird nur zum Schreiben der Fehlermeldung geöffnet (Modus append) und danach wieder geschlossen.

Die Datei kann mit jedem Editor gelesen werden!

W **Windows-Systeme**

W Die Logging-Datei hat den Namen `UPICL`*tid*`.UPL`, wobei tid die Thread-ID ist. In welchem Dateiverzeichnis die Logging-Datei abgelegt wird, können Sie mit der Umgebungsvariablen UPICLOG unter dem Registry Key UPICW32DLL oder in der Datei `UPIC.INI` (siehe Abschnitt „Ablaufumgebung, Umgebungsvariablen“ auf Seite 322f) festlegen.

W Wird die Umgebungsvariable UPICLOG nicht gesetzt, dann werden nacheinander (in der angegebenen Reihenfolge) ausgewertet:

- W Registry Key UPICW32DLL
- W die Datei UPIC.INI (nicht bei INI-File Mapping)
- W die Variable TEMP
- W die Variable TMP

Falls ein entsprechender Eintrag gefunden wird, wird das dort angegebene Verzeichnis genommen. Wird nichts gefunden, dann wird die Datei im Dateiverzeichnis \USR\TMP abgelegt. Dieses Verzeichnis muss vorhanden sein und das CPI-C-Programm muss die Schreibberechtigung für dieses Verzeichnis haben, sonst gehen die Logging-Dateien verloren.

Unix-Systeme

Der Name der Logging-Datei ist `UPICL`*pid*, wobei *pid* die Pozess-ID ist. In welchem Dateiverzeichnis die Logging-Datei abgelegt wird, legen Sie mit der Shellvariable UPICLOG fest. Ist die Shellvariable nicht gesetzt, wird die Datei im Dateiverzeichnis `/usr/tmp` abgelegt.

BS2000/OSD

Der Name der Logging-Datei ist UPICL*tsn*, dabei ist *tsn* die TSN der BS2000-Task.

Über die Jobvariable mit dem Linknamen UPICLOG legen Sie den Präfix der Logging-Datei fest (siehe Abschnitt „Ablaufumgebung, Binden, Starten“ auf Seite 319).
Ist UPICLOG nicht gesetzt, dann wird folgende Logging-Datei geschrieben:

```
##.usr.tmp.UPICLtsn
```

Wird im BS2000 ein UPIC-Prozess ohne vorheriges LOGOFF/LOGON neu gestartet, dann bleibt die TSN-Nummer *tsn* erhalten. Dadurch wird die Logging-Datei überschrieben!

7.4.2 UPIC-Trace

Beim Trägersystem UPIC ist es möglich, Verfolgerinformation für sämtliche CPI-C-Schnittstellenaufrufe zu erzeugen. Dies steuern Sie durch das Setzen der Variablen UPICTRACE.

Beim Aufruf *Enable_UTM_UPIC* wird der Inhalt der Variable ausgewertet. Falls sie gesetzt ist, werden beim Aufruf jeder Funktion die Parameter und die Benutzerdaten bis zu einer Länge von 128 Bytes Prozess-spezifisch in einer Datei protokolliert.
Beim *Disable_UTM_UPIC*-Aufruf wird die Protokollierung wieder ausgeschaltet.

Falls ein CPI-C-Aufruf einen Returncode ungleich CM_OK oder CM_DEALLOCATED_ABEND liefert, wird auch diese Fehlerursache in die UPIC-Trace-Datei protokolliert. Sie gibt bei der Fehlersuche detaillierte Hinweise zu einem speziellen Returncode.

UPIC-Trace einschalten

Den UPIC-Trace schalten Sie ein, indem Sie die Variable UPICTRACE entsprechend setzen. Der UPIC-Trace wird auf den einzelnen Plattformen wie folgt eingeschaltet:

W
- *Windows-Systeme:*

  W W W W W W W
  Der UPIC-Trace kann auf verschiedene Arten eingeschaltet werden:
  - indem die Umgebungsvariable UPICTRACE entsprechend gesetzt wird.
  - indem unter dem Registry Key UPICW32DLL der Value UPICTRACE die entsprechenden Daten erhält.
  - indem ein entsprechender Eintrag in der Datei `UPIC.INI` gemacht wird (siehe Seite 322). Der neue Wert ist sofort wirksam, d.h. ab dem folgenden Start eines CPI-C-Programms wird protokolliert.

  W W W W W W W
  Wenn die Umgebungsvariable UPICTRACE gesetzt ist, wird der Wert der Umgebungsvariable verwendet.
  Wenn die Umgebungsvariable nicht gesetzt ist, wird in der Registry geprüft, ob unter dem Key UPICW32DLL der Value UPICTRACE existiert und sein Datenfeld entsprechend ausgefüllt ist.
  Ist weder die Umgebungsvariable gesetzt noch der Registry Eintrag UPICTRACE vorhanden, dann wird, falls vorhanden, der Eintrag in der Datei `UPIC.INI` ausgewertet.

  W
  Für UPICTRACE kann folgendes gesetzt werden:

  W
  ```
  UPICTRACE=-S[X] [-r wrap] [-dpfadname]
  ```

X
- *Unix-Systeme:*

  X X
  Der UPIC-Trace wird eingeschaltet, wenn die Umgebungsvariablen UPICTRACE wie folgt gesetzt wird:

  X X
  ```
  UPICTRACE=-S[X] [-r wrap] [-dpfadname]
  export UPICTRACE
  ```

B
- *BS2000/OSD*:

  B
  Der UPIC-Trace wird wie folgt eingeschaltet:

  B B B
  ```
  /SET-JV-LINK LINK-NAME=*UPICTRA,JV-NAME=UPICTRACE
  /MODIFY-JV JV[-CONTENTS]=UPICTRACE,SET-VALUE='-S[X] [-r wrap]
  [-Dpfadname]'
  ```

  B
  Die Option -D muss hier als Großbuchstabe angegeben werden.

Die Optionen haben folgende Bedeutung:

-S Es erfolgt eine ausführliche Protokollierung der CPI-C-Aufrufe, ihrer Argumente und der Benutzerdaten in der maximalen Länge von 128 Bytes (Pflichtangabe).

-SX Es werden zusätzlich interne Informationen an der Schnittstelle zum Transportsystem protokolliert (siehe auch „Erweiterter UPIC-Trace“ auf Seite 342). Es wird empfohlen, immer diese Option zu verwenden, da Probleme häufig mit der Transportschnittstelle zusammenhängen.

Der Schalter -SX ist bei CMX/PCMX eine Erweiterung zum Schalter -S.
Bei der Kommunikation mit Socket hat dieser Schalter keine zusätzliche Wirkung zum Schalter -S.

-r *wrap*

Durch die Dezimalzahl *wrap* wird die maximale Größe der temporären Trace-Datei bestimmt.

Maximalwert von *wrap*: 128
Standardwert von *wrap*: 128

-d*pfadname* / -D

Der Pfadname kann mit Leerzeichen angegeben werden. Wenn Leerzeichen verwendet werden, muss der Pfadname in doppelte Hochkommata eingeschlossen werden. Sind keine Leerzeichen im Pfadnamen, können doppelte Hochkommata auch verwendet werden.

Windows-Systeme:
Die Trace-Dateien werden in dem mit *pfadname* angegebenen Dateiverzeichnis eingerichtet.
Wenn Sie `-d`*pfadname* nicht angeben, werden die Trace-Dateien in das Verzeichnis geschrieben, das in der Variablen TEMP angegeben ist. Ist TEMP nicht gesetzt, wird dasselbe mit TMP versucht. Sind beide Variablen nicht gesetzt, werden die Trace-Dateien im Dateiverzeichnis `\USR\TMP` eingerichtet. Dieses Verzeichnis muss dann vorhanden sein und das CPI-C-Programm muss in diesem Dateiverzeichnis schreibberechtigt sein, sonst gehen die Trace-Daten verloren.





Unix-Systeme:
Die Trace-Dateien werden in dem mit *pfadname* angegebenen Dateiverzeichnis eingerichtet.
Wenn Sie `-d`*pfadname* nicht angeben, werden die Trace-Dateien im Dateiverzeichnis `/usr/tmp` eingerichtet. Das CPI-C-Programm muss in diesem Dateiverzeichnis schreibberechtigt sein, sonst gehen die Trace-Daten verloren.

BS2000/OSD:
Für die Trace-Dateien wird ein Datei-Präfix angegeben, dass keine Leerzeichen haben sollte.
Wenn Sie `-D` nicht angeben, werden die Namen der Trace-Dateien um das Präfix `##.usr.tmp.` erweitert. Die Trace-Dateien werden unter der Kennung abgelegt, unter der das Programm gestartet wird. Das CPI-C-Programm muss die Datei öffnen können, sonst gehen die Trace-Daten verloren.

Beispiel

Bei Angabe von `-DTRC` wird die Trace-Datei *TRC.UPICTtsn* geschrieben.

Trace-Dateien

Die Verfolgerinformation wird in einer temporären Datei abgelegt. Diese Datei wird beim *Enable_UTM_UPIC*-Aufruf eingerichtet. Sie bleibt bis zum *Disable_UTM_UPIC*-Aufruf geöffnet. Die maximale Größe dieser temporären Datei bestimmen Sie durch die Dezimalzahl *wrap*.

In die Datei wird solange protokolliert, bis der Wert (*wrap* `* BUFSIZ`) Bytes (BUFSIZ wie in `stdio.h)` überschritten wird. Dann wird eine zweite temporäre Datei angelegt, die genauso behandelt wird.

Jedesmal, wenn der Wert (*wrap* `* BUFSIZ`) Bytes in der aktuellen Datei überschritten wird, schaltet der Verfolger auf die andere Datei um. Der alte Inhalt dieser Datei wird dabei überschrieben.

Die Dateinamen der Trace-Dateien sind Plattform-spezifisch. Folgende Dateinamen werden vergeben:

Name der	**Windows-Systeme**	**Unix-Systeme**	**Unix-Systeme,** wenn Threads in Programmen verwendet werden	**BS2000**
1. Datei	UPICT*tid*[*].upt	UPICT*pid*[†]	UPICT*pid*[2].*tid*[1]	UPICT*tsn*[‡]
2. Datei	UPICU*tid*[1].upt	UPICU*pid*[2]	UPIUT*pid*[2].*tid*[1]	UPICU*tsn*[3]

[*] tid = Thread ID

[†] pid = Process ID

[‡] tsn = TSN-Nummer

Erweiterter UPIC-Trace

Beim erweiterten UPIC-Trace werden zusätzlich interne Informationen an der Schnittstelle zum Transportsystem (UPIC <-> CMX bzw. PCMX) protokolliert. Zusätzlich zu den UPIC-Aufrufen werden die zugehörigen CMX-Aufrufe protokolliert.

Das erweiterte Protokoll ist wie folgt aufgebaut:

Nach der Protokollierung eines UPIC-Aufrufs wird zunächst eine Zeile mit einem ergänzenden Klartext ausgegeben. Danach folgt in zwei Zeilen die Protokollierung der zuletzt aufgerufenen CMX-Funktionen. Die Informationen sind durch Komma bzw. <newline> getrennt.

1. Zeile:

Die erste Zeile enthält folgende Informationen:

- Name der aufgerufenen CMX-Funktion.
- Returncode der CMX-Funktion *t_error*. Der Returncode ist eine hexadezimale Zahl. Ist diese von Null verschieden, dann können Sie ihr die Ursache eines aufgetretenen Fehlers entnehmen.

  Die Hexadezimalzahl kann wie folgt decodiert werden:

  – mit dem Kommando `cmxdec -d 0x`*hexadezimalzahl* oder

  – mit Hilfe des Windows-Programms **Trace Control** im Programm-Fenster PCMX. Wählen Sie im Menü **Options** den Befehl **Error Decoding** aus.
- Returncode der CMX-Funktion als Dezimalzahl (falls die CMX-Funktion einen *int*-Wert zurückliefert).

Eine wichtige Ausnahme bildet die CMX-Funktion *t_event*. Ihr Rückgabewert (d.h. das aufgetretene Ereignis) wird immer an erster Stelle der zweiten Zeile ausgegeben.

2. Zeile:

Die zweite Zeile protokolliert einen CMX-Aufruf, der aufgrund eines eingetroffenen Ereignisses (*t_event*) im Zusammenhang mit der in der 1. Zeile protokollierten CMX-Funktion aufgerufen wurde. Die 2. Zeile enthält nacheinander folgende Informationen:

– Name des Ereignisses, das die Funktion *t_event* zurückgeliefert hat.

– Name der aufgerufenen CMX-Funktion.

– Returncode von *t_error*, falls bei der zweiten CMX-Funktion ein Fehler auftrat. Er gibt gegebenenfalls den Grund für einen Verbindungsabbau an. Die Zahl kann wie oben beschrieben mit `cmxdec` decodiert werden. Der Wert „-1“ besagt, dass kein Verbindungsabbaugrund vorliegt.

– Hinter dem letzten Komma dieser Zeile kann ein UPIC-Returncode folgen.

Wurde im Zusammenhang mit der in der 1. Zeile protokollierten CMX-Funktion keine weitere CMX-Funktion aufgerufen, dann wird in der 2. Zeile nur ein Blank und eine Null ausgegeben.

UPIC-Trace ausschalten

Der UPIC-Trace wird ausgeschaltet, indem die Variable UPICTRACE ohne Parameter gesetzt wird:

- *Windows-Systeme:*
  - indem Sie das folgende Set-Kommando absetzen:

    ```
    SET UPICTRACE=
    ```

  - indem Sie in der Datei `UPIC.INI` den Eintrag für UPICTRACE wie folgt ändern:

    ```
    UPICTRACE=
    ```

    Der Trace wird dann beendet, sobald das CPI-C-Programm beendet wird.
  - indem sie den Wert UPICTRACE unter dem Key UPICW32DLL löschen oder lediglich das Datenfeld vom Wert UPICTRACE leeren.
- *Unix-Systeme:*

  ```
  UPICTRACE=
  export UPICTRACE
  ```

- *BS2000/OSD:*
  - mit dem Kommando
    `/MODIFY-JV JV[-CONTENTS]=UPICTRACE,SET-VALUE=''`
    Der Inhalt der JV wird gelöscht.
  - mit dem Kommando `/DELETE-JV`
    Die komplette JV wird gelöscht.

  Bei Neustart eines UPIC-Prozesses ist der Trace ausgeschaltet.

UPIC-Trace aufbereiten

Die Verfolgerinformation liegt bereits in abdruckbarer Form vor, sie muss deshalb nicht mehr durch ein Dienstprogramm aufbereitet werden.

Jede Aktion wird mit der entsprechenden Uhrzeit und den übertragenen Werten protokolliert.

7.4.3 PCMX-Diagnose (Windows-Systeme)

Die PCMX-Diagnose wird durch das Programm `cmxtrc32.exe` gesteuert. Dieses Programm wird in der Windows-Programmgruppe PCMX-32 durch Doppelklick auf das Symbol „Trace Control“ aufgerufen. Mit diesem Programm können Sie

- PCMX-Traces einschalten und ausschalten
- PCMX-Traces am Bildschirm anschauen oder ausdrucken
- PCMX-Fehlercodes decodieren (Option „Error Decoding“)

Wie dieses Programm arbeitet, ist in der Online-Hilfe der PCMX-Programmgruppe genauer beschrieben.

8 Beispiele

Dieses Kapitel enthält Hinweise auf die mit ausgelieferten Beispielprogramme sowie einfache Generierungsbeispiele für eine Kopplung einer CPI-C-Anwendung auf Windows-Systemen mit UTM-Anwendungen auf BS2000/OSD, auf Unix- und Windows-Systemen.

8.1 Programmbeispiele für Windows-Systeme

Mit dem openUTM-Client für Trägersystem UPIC werden folgende Programmbeispiele ausgeliefert:

uptac	Komplettes CPI-C-Anwendungsprogramm.
utp32	Programm für die interaktive Eingabe einzelner CPI-C-Aufrufe.
tpcall	Komplettes XATMI-Programm.
upic-cob	Ein Cobol-Projekt.
UpicSimpleClient	Komplettes CPI-C Programm in C++.

Zusätzlich wird die Local Definition File *tpcall.ldf* ausgeliefert, aus der das Tool XTGEN32 bzw. XATMIGEN eine Local Configuration File für das XATMI-Programm *tpcall* erzeugt.

uptac, *utp32*, *tpcall* sind nach kurzer Vorbereitung ablauffähig. Sie werden z.B. durch Doppelklick auf entsprechende Symbole aufgerufen, die nach der Installation im Programmfenster **openUTM-Client** zu finden sind.

Alle Client-Programmbeispiele sind darauf abgestimmt, mit der openUTM-Beispiel-Anwendung auf der Server-Seite zu kommunizieren. Näheres dazu finden Sie in der Readme-Datei zur openUTM-Beispiel-Anwendung.

Die folgenden Abschnitte stellen diese Programmbeispiele kurz vor und beschreiben die zum Ablauf notwendigen Vorbereitungen.

8.1.1 uptac

uptac ist ein einfaches CPI-C-Anwendungsprogramm. Es besteht aus den in der folgenden Tabelle aufgeführten Dateien. Die Dateien befinden sich nach der Installation im Verzeichnis *upic-dir*`\samples`.

Dateiname	**Art der Datei**
uptac32.c	C-Source-Code des Programms; kann ausgedruckt werden
uptac32.dsp	Project File von Microsoft Visual C++ zum Erzeugen einer „.exe“
uptac32.exe	Ausführbares Programm uptac
uptac32.bat	Batchdatei für uptac32.exe

Damit *uptac* mit der openUTM-Beispiel-Anwendung kommunizieren kann, müssen Sie UPIC konfigurieren, z.B. können in der `upicfile` und gegebenenfalls in der TNS-Datenbasis folgende Einträge vorhanden sein, siehe Mustereinträge in den ausgelieferten Dateien *upic-dir*\UPICFILE.SMP und *upic-dir*\`Tnsentry.smp`:

Side Information-Datei:

```
LN.DEFAULT UPIC0000
SD.DEFAULT SMP30111.unixhost PORT=30111
```

TNS-Eintrag (kann erstellt werden, ist aber nicht mehr nötig):

```
UPIC0000\
      TSEL RFC1006 T'UPIC0000'                    ; local name TNS
SMP30111.unixhost\
      TA RFC1006 unixhost PORT 30111 T'SMP30111'     ; partner_LU_name TNS
```

unixhost ist der symbolische Name des Rechners, auf dem die openUTM-Beispiel-Anwendung läuft. Falls *uptac* mit einer anderen UTM-Anwendung (z.B. im BS2000) kommunizieren soll, müssen Sie alle Einträge außer `LN.DEFAULT` entsprechend anpassen.

In der Transportadresse (TA...) können Sie anstelle des symbolischen Namens auch die Internet-Adresse des Unix-Systems angeben. Überprüfen Sie dabei bitte, ob die Portnummer „30111“ und der T-Selektor „SMP30111“ auch auf der Server-Seite eingetragen sind.

8.1.2 utp32

W W W W *utp32* ist ein Beispiel für eine Visual Basic-Client-Anwendung. Mit ihr können Sie die Kommunikation über die CPI-C-Schnittstelle schrittweise abwickeln, indem Sie interaktiv einzelne CPI-C-Aufrufe mit ihren Parametern in ein Dialogfeld eintragen. Sie erhalten dabei den zugehörigen Returncode des Aufrufs.

8.1.3 tpcall

W W W W *tpcall* ist ein einfaches XATMI-Anwendungsprogramm, mit dem ein synchroner Request/Response mit der openUTM-Beispiel-Anwendung realisiert werden kann. *tpcall* besteht aus den in der folgenden Tabelle aufgelisteten Dateien, die sich nach Installation im Verzeichnis *xatmi-dir*\Samples befinden.

Dateiname	Art der Datei
tpcall.c	C-Source-Code des Programms; kann ausgedruckt werden
tpcall_32.dsp	Project File von Microsoft Visual C++ zum Erzeugen einer „.exe“
tpcall32.exe	ausführbares Programm tpcall

W Bevor Sie per *tpcall* mit der Beispiel-Anwendung kommunizieren können, müssen Sie

W W – wie bei *uptac* die Einträge in der upicfile und im TNS erzeugen, siehe Abschnitt „uptac“ auf Seite 346,

W W – eine Local Configuration File erzeugen, indem Sie das Symbol XATMIGEN bzw. XTGEN32 anklicken, das sich im Programmfenster **openUTM-Client** befindet.

W W Es wird dann aus der mit ausgelieferten Local Definition File *xatmi-dir*\Samples\tpcall.ldf die Datei xatmilcf (im selben Verzeichnis) erzeugt.

W W W Falls *tpcall* mit anderen Anwendungen kommunizieren soll, müssen Sie ggf. die upicfile und damit auch die Local Definition File tpcall.ldf anpassen (Anweisung SVCU ... DEST, siehe auch Abschnitt „UPIC konfigurieren“ auf Seite 278).

8.1.4 upic-cob

W W Das Verzeichnis enthält ein Beispielprojekt zum Erstellen einer UPIC-Cobol-Anwendung. Das Beispiel ist unter einem Cobol-Compiler von MicroFocus entworfen worden.

8.1.5 UpicSimpleClient



Das Verzeichnis UpicSimpleClient enthält ein C++-Programm. Sie finden dort die ausführbare Datei *UpicSimpleClient.exe* sowie alle notwendigen Dateien, um selbst die Datei *UpicSimpleClient.exe* erstellen zu können.

8.2 Generierung UPIC auf Windows-System <-> openUTM auf BS2000/OSD

Das folgende Generierungsbeispiel erläutert das Prinzip, wie die Anbindung einer CPI-C-Anwendung in Windows-Systemen an openUTM auf BS2000/OSD generiert werden muss. Dabei wird die Kopplung über RFC1006 dargestellt.

Das Windows-System hat im Beispiel den symbolischen Hostnamen HOST123, der BS2000-Rechner den Namen HOST456.

Die TNS-Generierung ist nur noch zum Vergleich dargestellt, da sie nicht mehr benötigt wird.

8.2.1 Generierung auf dem Windows-System

UPIC-Parameter

```
Enable_UTM_UPIC "UPICTTY"
Initialize_Conversation "sampladm"
```

Side Information Datei C:\UPIC\UPICFILE

```
* UTM(BS2000) Anwendung
SDsampladm UTMUPICR.HOST456 KDCHELP
* oder, falls automatische Konvertierung der Benutzerdaten
* gewünscht wird
HDsampladm UTMUPICR.HOST456 KDCHELP
```

TNS-Einträge im tnsxfrm-Format

```
UPICTTY\
        TSEL RFC1006 T'UPICTTY'                   ; local name RFC1006
UTMUPICR.HOST456\
        TA RFC1006 HOST456 PORT 102  T'UTMUPICR' ; partner name RFC1006
```

8.2.2 Generierung auf dem BS2000-Rechner

Im Beispiel ist HOST123 der Name des PCs als entferntes System, der statisch in der BCAM-RDF (*resource-definition-file*) oder dynamisch per BCIN eingetragen sein muss.

KDCDEF-Generierung für die UTM-Anwendung auf BS2000/OSD

```
BCAMAPPL UTMUPICR, T-PROT=ISO

PTERM    UPICTTY, PTYPE=UPIC-R, LTERM=UPIC,
                  BCAMAPPL=UTMUPICR, PRONAM=HOST123
LTERM    UPIC, USER=UPICUSER
USER     UPICUSER, STATUS=ADMIN
```

8.3 Generierung UPIC auf Windows-System <-> openUTM auf Unix-System

Das folgende Generierungsbeispiel erläutert das Prinzip, wie die Anbindung einer CPI-C-Anwendung in Windows-Systemen an openUTM auf Unix-Systemen generiert werden muss. Dabei wird die Kopplung über RFC1006 dargestellt.

Das Windows-System hat im Beispiel den symbolischen Hostnamen HOST123, das Unix-System den Namen HOST789.

Die TNS-Generierung ist nur noch zum Vergleich dargestellt, da sie nicht mehr benötigt wird.

8.3.1 Generierung auf dem Windows-System

UPIC-Parameter

```
Enable_UTM_UPIC "UPIC0000"
Initialize_Conversation "sampladm"
```

Side Information Datei C:\UPIC\UPICFILE

```
* UPIC-Anwendung auf dem Windows-System
LNUPIC0000 UPICTTY
```

```
* partner RFC1006
SDsampladm UTMUPICR.HOST789 KDCHELP PORT=1230
```

TNS-Einträge im tnsxfrm-Format

Adressformat RFC1006:

```
UPICTTY\
        TSEL RFC1006 T'UPICTTY'                    ; local name RFC1006
        TSEL LANINET A'4711'                       ; local name
UTMUPICR.HOST789\
        TA RFC1006 HOST789 PORT 1230  T'UTMUPICR' ; partner RFC1006
```

HOSTS-Datei

In der Datei *win-dir*`\HOSTS` wird `HOST789` auf die Internetadresse abgebildet:

```
internetadresse HOST789
```

win-dir steht dabei für das Installationsverzeichnis von Windows, z.B. `C:\winnt\system32\drivers\etc.`

8.3.2 Generierung auf dem Unix-System

X **KDCDEF-Generierung für die UTM-Anwendung auf dem Unix-System**

```
BCAMAPPL UTMUPICR

PTERM    UPICTTY, PTYPE=UPIC-R, LTERM=UPIC,
                  BCAMAPPL=UTMUPICR, PRONAM=HOST123
LTERM    UPIC, USER=UPICUSER
USER     UPICUSER, STATUS=ADMIN
```

9 Anhang

Der Anhang enthält:

- Unterschiede zur X/Open-Schnittstelle CPI-C
- Zeichensatztabellen
- Zustandstabellen

9.1 Unterschiede zur X/Open-Schnittstelle CPI-C

Der Abschnitt beschreibt für CPI-C mit Trägersystem UPIC alle Erweiterungen und Besonderheiten gegenüber der X/Open-Schnittstelle CPI-C.

Erweiterungen gegenüber CPI-C

- Es werden folgende zusätzliche UPIC-spezifische Funktionen angeboten:

  Enable_UTM_UPIC
  Extract_Client_Context
  Extract_Communication_Protocol
  Extract_Conversation_Encryption_Level
  Extract_Cursor_Offset
  Extract_Convertion
  Extract_Secondary_Return_Code
  Extract_Shutdown_State
  Extract_Shutdown_Time
  Extract_Transaction_State
  Disable_UTM_UPIC
  Set_Allocate_Timer
  Set_Client_Context
  Set_Communication_Protocol
  Set_Conversation_Encryption_Level
  Set_Conversation_New_Password
  Set_Convertion
  Set_Function_Key
  Set_Partner_Host_Name
  Set_Partner_IP_Adress

Set_Partner_Port
Set_Partner_Tsel
Set_Partner_Tsel_Format
Set_Receive_Timer
Specify_Local_Port
Specify_Local_Tsel
Specify_Local_Tsel-Format
Specify_Secondary_Return_Code

Die Funktionen *Enable_UTM_UPIC* und *Disable_UTM_UPIC* regeln das An- und Abmelden von CPI-C-Programmen beim Trägersystem UPIC. Ohne die Verwendung dieser beiden Aufrufe ist eine Anbindung an eine UTM-Anwendung nicht möglich. Genaueres hierzu finden Sie im Abschnitt „CPI-C-Aufrufe bei UPIC“ auf Seite 96 und im Kapitel „Konfigurieren“ auf Seite 291.

- Bei UPIC werden die Aufrufe *Send_Mapped_Data* und *Receive_Mapped_Data* verwendet, um Formatnamen zu senden und zu empfangen.
- Automatische Konvertierung der Benutzerdaten per Konfigurierung

  Dadurch besteht zusätzlich die Möglichkeit der automatischen Code-Umsetzung von Benutzerdaten zwischen ASCII- und EBCDIC-Code, siehe auch Abschnitt „Code-Konvertierung“ auf Seite 68. Zum einen wird dadurch der Aufwand bei der Erstellung einer Anwendung reduziert. Zum anderen wird die Möglichkeit geschaffen, mit einem einzigen CPI-C-Programm sowohl mit einer UTM-Anwendung auf einem Unix-System auf Basis des ASCII-Codes als auch mit einer UTM-Anwendung auf BS2000/OSD auf Basis des EBCDIC-Codes zu kommunizieren (falls die Benutzerdaten keine Binärinformation enthalten, die bei der Codeumsetzung verfälscht würde).

Besonderheiten der CPI-C-Implementierung

- Der Name für *partner_LU_name* darf höchstens 32 Zeichen lang sein; bei lokaler Anbindung über UPIC-Local (Unix-, Windows-System) sogar nur bis zu 8 Zeichen.
- Der Name für *TP_name* darf höchstens acht Zeichen lang sein.

Migration von X/Open CPI-C Version 1 nach X/Open CPI-C Version 2

Die X/Open CPI-C-Spezifikation Version 2 enthält einige Änderungen gegenüber der vorangegangenen CPI-C-Version. Diese Änderungen wirken sich auch auf CPI-C-Programme mit Trägersystem UPIC aus, da sie übernommen wurden.

Folgende Änderungen betreffen CPI-C-Anwendungen in C:

- CPI-C-Version 2 definiert alle Funktionen vom Typ `void`. Programme, die den Returncode prüfen, müssen den Parameter CM_RETURN_CODE abfragen.

- Einige Parameter haben in X/Open CPI-C Version 2 andere Typen als vorher. Einige Compiler könnten Warnings ausgeben, falls bestehende CPI-C-Programme mit dem neuen CPI-C Version 2 Includefile übersetzt werden.

  Die folgende Tabelle gibt einen Überblick.

Parameter	**Original X/Open CPI-C**	**X/Open CPI-C Version 2**
Conversation ID Parameter	Conversation_ID (char [8])	unsigned char CM_PTR (unsigned char *)
Character pointers	char *	unsigned char CM_PTR (unsigned char *)
Length parameters	int *	CM_INT32 CM_PTR (signed long int *)
Definition of return codes and numeric Parameters	typedef enum	#define

Tabelle 15: Geänderte Parameter bei X/Open Version 2

Bestehende CPI-C-Programme sind objektcode-kompatibel mit openUTM-Client V6.1 einsetzbar. Um bestehende Programme ohne Sourcecode-Änderungen (bedingt durch den Übergang von X/Open Version1 zu X/Open Version 2) nutzen zu können, bietet das Trägersystem UPIC folgendes an:

– Die Include-Datei enthält spezielle auf die CPI-C-Schnittstelle von UPIC zugeschnittene *#defines*.

– Beim Compilieren müssen Sie spezielle Compiler-Optionen (Präprozessor-Symbole) setzen.

  Durch die Compiler-Option *UTM_UPIC_V11* wird der X/Open-konforme Teil des Include Files abgeschaltet und die alten Definitionen (d.h. ohne die Security-Funktionen *Prepare_To_Receive*, *Set_Receive_Timer*, *Set_Function_Key*, *Send_Mapped_Data*, *Receive_Mapped_Data* und *Set_Receive_Type*) aktiviert. Ohne diese Compiler-Option gilt das Umgekehrte.

  Unix-Systeme:
  Bei 64-Bit-Systemen darf die Compiler-Option UTM_UPIC_V11 nicht gesetzt werden.

  Windows-Systeme:
  Beim Compilieren von CPI-C-Programmen auf Windows-Systemen müssen Sie die Compiler-Optionen *UTM_ON_WIN32* unbedingt setzen. Die Wirkung dieser Option können Sie der Include-Datei *UPIC.H* entnehmen. Sie befindet sich im Verzeichnis *upic-dir*`\include`. *upic-dir* ist das Installationsverzeichnis von openUTM-Client mit Trägersystem UPIC.



Achten Sie darauf, dass die Compiler-Option *UTM_UPIC_V11* **nicht** zusammem mit der Compiler-Option *UTM_ON_WIN32* verwendet werden darf. Werden diese beiden Compiler-Optionen zusammen gesetzt, dann ist das Programm nicht ablauffähig.

– Die Funktionsprototypen werden für ANSI- und für K&R-Compiler angeboten. Das übliche __STDC__ schaltet ANSI ein.

Bestehende CPI-C-Programme, die nach der CPI-C Version 1 codiert wurden, sind objektcode-kompatibel mit openUTM-Client V6.1. Diese Kompatibilität wird für zukünftige Versionen nicht mehr gewährleistet.

9.2 Zeichensätze

An der Schnittstelle CPI-C darf der Inhalt der Variable *sym_dest_name* nur aus Zeichen eines vorgegebenen Zeichenvorrats bestehen.

Im folgenden werden die Zeichensätze und ihre Zuordnung zu den Variablen beschrieben.

Variable	Zeichensatz
sym_dest_name	Set 1

Zeichen	Zeichensatz	
	Set 1	**Set 2**
.		X
<		X
(		X
+		X
&		X
*		X
)		X
;		X
-		X
/		X
,		X
%		X
-		X
>		X
?		X
:		X
'		X
=		X
"		X
a-z	X	X
A-Z	X	X
0-9	X	X

Tabelle 16: Zeichensätze

T.61-Zeichensatz

<table>
<tr><th></th><th>0</th><th>1</th><th>2</th><th>3</th><th>4</th><th>5</th><th>6</th><th>7</th><th>8</th><th>9</th><th>...</th><th>F</th></tr>
<tr><td>0</td><td></td><td></td><td>SP</td><td>0</td><td>@</td><td>P</td><td></td><td>p</td><td></td><td></td><td></td><td></td></tr>
<tr><td>1</td><td></td><td></td><td>!</td><td>1</td><td>A</td><td>Q</td><td>a</td><td>q</td><td></td><td></td><td></td><td></td></tr>
<tr><td>2</td><td></td><td></td><td>"</td><td>2</td><td>B</td><td>R</td><td>b</td><td>r</td><td></td><td></td><td></td><td></td></tr>
<tr><td>3</td><td></td><td></td><td>#</td><td>3</td><td>C</td><td>S</td><td>c</td><td>s</td><td></td><td></td><td></td><td></td></tr>
<tr><td>4</td><td></td><td></td><td>¤</td><td>4</td><td>D</td><td>T</td><td>d</td><td>t</td><td></td><td></td><td></td><td></td></tr>
<tr><td>5</td><td></td><td></td><td>%</td><td>5</td><td>E</td><td>U</td><td>e</td><td>u</td><td></td><td></td><td></td><td></td></tr>
<tr><td>6</td><td></td><td></td><td>&</td><td>6</td><td>F</td><td>V</td><td>f</td><td>v</td><td></td><td></td><td></td><td></td></tr>
<tr><td>7</td><td></td><td></td><td>´</td><td>7</td><td>G</td><td>W</td><td>g</td><td>w</td><td></td><td></td><td></td><td></td></tr>
<tr><td>8</td><td>BS</td><td></td><td>(</td><td>8</td><td>H</td><td>X</td><td>h</td><td>x</td><td></td><td></td><td></td><td></td></tr>
<tr><td>9</td><td></td><td>SS2</td><td>)</td><td>9</td><td>I</td><td>Y</td><td>i</td><td>y</td><td></td><td></td><td></td><td></td></tr>
<tr><td>A</td><td>LF</td><td>SUB</td><td>*</td><td>:</td><td>J</td><td>Z</td><td>j</td><td>z</td><td></td><td></td><td></td><td></td></tr>
<tr><td>B</td><td></td><td>ESC</td><td>+</td><td>;</td><td>K</td><td>[</td><td>k</td><td></td><td>PLD</td><td>CSI</td><td></td><td></td></tr>
<tr><td>C</td><td>FF</td><td></td><td>,</td><td><</td><td>L</td><td></td><td>l</td><td>|</td><td>PLU</td><td></td><td></td><td></td></tr>
<tr><td>D</td><td>CR</td><td>SS3</td><td>-</td><td>=</td><td>M</td><td>]</td><td>m</td><td></td><td></td><td></td><td></td><td></td></tr>
<tr><td>E</td><td>LS1</td><td></td><td>.</td><td>></td><td>N</td><td></td><td>n</td><td></td><td></td><td></td><td></td><td></td></tr>
<tr><td>F</td><td>LS0</td><td></td><td>/</td><td>?</td><td>O</td><td>-</td><td>o</td><td></td><td></td><td></td><td></td><td></td></tr>
</table>

Tabelle 17: Codetabelle T.61 gemäß CCITT Recommodation

Bedeutung der Abkürzungen:

BS=	BACKSPACE	SUB=	SUBSTITUTE CHARACTER
LF=	LINE FEED	ESC=	ESCAPE
FF=	FORM FEED	SS3=	SINGLE-SHIFT THREE
CR=	CARRIAGE RETURN	SP=	SPACE
LS1=	LOCKING SHIFT ONE	PLD=	PARTIAL LINE DOWN
LS0=	LOCKING SHIFT ZERO	PLU=	PARTIAL LINE UP
SS2=	SINGLE-SHIFT TWO	CSI=	CONTROL SEQUENCE INTRODUCER

Tabelle 18: Abkürzungen für Sonderzeichen

9.3 Zustandstabelle

Die folgende Tabelle gibt für die einzelnen Aufrufe (abhängig von deren Ergebnis) den Folgezustand des Programms an, falls es vorher in einem bestimmten Zustand war. Die Bedeutung der in der Tabelle verwendeten Abkürzungen werden im Anschluss erklärt.

Aufruf	**Ergebnis**	**Folgezustand, falls vorher im Zustand**				
		Start	**Reset**	**Init.**	**Send**	**Receive**
Initialize_Conversation	ok	psc	Init.	psc	psc	psc
Initialize_Conversation	pc	psc	-	psc	psc	psc
Initialize_Conversation	ps	psc	-	psc	psc	psc
Allocate	ok	psc	psc	Send	psc	psc
Allocate	ae	psc	psc	Reset	psc	psc
Allocate	pc	psc	psc	-	psc	psc
Allocate	pe	psc	psc	-	psc	psc
Allocate	ps	psc	psc	-	psc	psc
Deallocate	ok	psc	psc	Reset	Reset	Reset
Deallocate	pc	psc	psc	-	-	-
Deallocate	ps	psc	psc	-	-	-
Deferred_Deallocate	-	-	-	-	-	-
Extract_Client_Context	ok	psc	-	-	-	-
Extract_Client_Context	pc	psc	-	-	-	-
Extract_Client_Context	ps	psc	-	-	-	-
Extract_Communication_Protocol	ok	psc	psc	-	psc	psc
Extract_Communication_Protocol	pc	psc	psc	-	psc	psc
Extract_Communication_Protocol	ps	psc	psc	-	psc	psc
Extract_Conversation_Encryption_Level	ok	psc	psc	-	-	-
Extract_Conversation_Encryption_Level	pc	psc	psc	-	-	-
Extract_Conversation_Encryption_Level	ps	psc	psc	-	-	-
Extract_Conversation_State	ok	psc	psc	-	-	-
Extract_Conversation_State	pc	psc	psc	-	-	-
Extract_Conversation_State	ps	psc	psc	-	-	-
Extract_Convertion	ok	psc	psc	-	psc	psc
Extract_Convertion	pc	psc	psc	-	psc	psc
Extract_Convertion	ps	psc	psc	-	psc	psc

Tabelle 19: Zustandstabelle für CPI-C-Aufrufe

Aufruf	Ergebnis	Folgezustand, falls vorher im Zustand				
		Start	Reset	Init.	Send	Receive
Extract_Cursor_Offset	ok	psc	-[*]	-	-	-
Extract_Cursor_Offset	pc	psc	-	-	-	-
Extract_Cursor_Offset	ps	psc	-	-	-	-
Extract_Partner_LU_Name	ok	-	-	-	-	-
Extract_Partner_LU_Name	pc	-	-	-	-	-
Extract_Partner_LU_Name	ps	-	-	-	-	-
Extract_Secondary_Information	ok	-	-	-	-	-
Extract_Secondary_Information	pc	-	-	-	-	-
Extract_Secondary_Information	ps	-	-	-	-	-
Extract_Secondary_Return_Code	ok	psc	psc	-	-	-
Extract_Secondary_Return_Code	nr	psc	psc	-	-	-
Extract_Secondary_Return_Code	pc	psc	psc	-	-	-
Extract_Secondary_Return_Code	ps	psc	psc	-	-	-
Extract_Shutdown_State	ok	psc	-[1]	psc	-	-
Extract_Shutdown_State	pc	psc	-[1]	psc	-	-
Extract_Shutdown_State	ps	psc	-[1]	psc	-	-
Extract_Shutdown_Time	ok	psc	-[1]	psc	-	-
Extract_Shutdown_Time	pc	psc	-[1]	psc	-	-
Extract_Shutdown_Time	ps	psc	-[1]	psc	-	-
Extract_Transaction_State	ok	psc	-[1]	psc	-	-
Extract_Transaction_State	pc	psc	-[1]	psc	-	-
Extract_Transaction_State	ps	psc	-[1]	psc	-	-
Prepare_To_Receive	ok	psc	psc	psc	Receive	-
Prepare_To_Receive	da	psc	psc	psc	Reset	psc
Prepare_To_Receive	pc	psc	psc	psc	-	psc
Prepare_To_Receive	rf	psc	psc	psc	Reset	psc
Receive / Receive_Mapped_Data	ok{dr,no}	psc	psc	psc	Receive	-
Receive / Receive_Mapped_Data	ok{nd,se}	psc	psc	psc	-	Send
Receive / Receive_Mapped_Data	ok{dr,se}	psc	psc	psc	-	Send
Receive / Receive_Mapped_Data	ae	psc	psc	psc	Reset	Reset
Receive / Receive_Mapped_Data	da	psc	psc	psc	Reset	Reset

Tabelle 19: Zustandstabelle für CPI-C-Aufrufe

Aufruf	**Ergebnis**	**Folgezustand, falls vorher im Zustand**				
		Start	**Reset**	**Init.**	**Send**	**Receive**
Receive / Receive_Mapped_Data	dn	psc	psc	psc	Reset	Reset
Receive / Receive_Mapped_Data	rf	psc	psc	psc	Reset	Reset
Receive / Receive_Mapped_Data	oi,un	psc	psc	psc	Receive	-
Receive / Receive_Mapped_Data	pc	psc	psc	psc	-	-
Receive / Receive_Mapped_Data	ps	psc	psc	psc	-	-
Send_Data / Send_Mapped_Data	ok	psc	psc	psc	-	psc
Send_Data / Send_Mapped_Data	ae	psc	psc	psc	Reset	psc
Send_Data / Send_Mapped_Data	da	psc	psc	psc	Reset	psc
Send_Data / Send_Mapped_Data	pc	psc	psc	psc	-	psc
Send_Data / Send_Mapped_Data	rf	psc	psc	psc	Reset	psc
Set_Allocate_Timer	ok	psc	psc	-	psc	psc
Set_Allocate_Timer	pc	psc	psc	-	psc	psc
Set_Allocate_Timer	ps	psc	psc	-	psc	psc
Set_Client_Context	ok	psc	psc	psc	-	psc
Set_Client_Context	pc	psc	psc	psc	-	psc
Set_Client_Context	ps	psc	psc	psc	-	psc
Set_Communication_Protocol	ok	psc	psc	-	psc	psc
Set_Communication_Protocol	pc	psc	psc	-	psc	psc
Set_Communication_Protocol	ps	psc	psc	-	psc	psc
Set_Conversation_Encryption_Level	ok	psc	psc	-	psc	psc
Set_Conversation_Encryption_Level	pc	psc	psc	-	psc	psc
Set_Conversation_Encryption_Level	ps	psc	psc	-	psc	psc
Set_Convertion	ok	psc	psc	-	psc	psc
Set_Convertion	pc	psc	psc	-	psc	psc
Set_Convertion	ps	psc	psc	-	psc	psc
Set_Conversation_Security_Type	ok	psc	psc	-	psc	psc
Set_Conversation_Security_Type	pc	psc	psc	-	psc	psc
Set_Conversation_Security_Type	pn	psc	psc	-	psc	psc
Set_Conversation_Security_New_Pass word	ok	psc	psc	-	psc	psc
Set_Conversation_Security_New_Pass word	pc	psc	psc	-	psc	psc

Tabelle 19: Zustandstabelle für CPI-C-Aufrufe

Aufruf	Ergebnis	Folgezustand, falls vorher im Zustand				
		Start	Reset	Init.	Send	Receive
Set_Conversation_Security_Password	ok	psc	psc	-	psc	psc
Set_Conversation_Security_Password	pc	psc	psc	-	psc	psc
Set_Conversation_Security_User_ID	ok	psc	psc	-	psc	psc
Set_Conversation_Security_User_ID	pc	psc	psc	-	psc	psc
Set_Deallocate_Type	ok	psc	psc	-	-	-
Set_Deallocate_Type	pc	psc	psc	-	-	-
Set_Deallocate_Type	ps	psc	psc	-	-	-
Set_Function_Key	ok	psc	psc	psc	-	-
Set_Function_Key	pc	psc	psc	psc	-	-
Set_Function_Key	ps	psc	psc	psc	-	-
Set_Receive_Timer	ok	psc	psc	psc	-	-
Set_Receive_Timer	pc	psc	psc	psc	-	-
Set_Receive_Timer	ps	psc	psc	psc	-	-
Set_Receive_Type	ok	-	-	-	-	-
Set_Receive_Type	pc	-	-	-	-	-
Set_Partner_Host_Name	ok	psc	psc	-	psc	psc
Set_Partner_Host_Name	pc	psc	psc	-	psc	psc
Set_Partner_Host_Name	ps	psc	psc	-	psc	psc
Set_Partner_IP_Address	ok	psc	psc	-	psc	psc
Set_Partner_IP_Address	pc	psc	psc	-	psc	psc
Set_Partner_IP_Address	ps	psc	psc	-	psc	psc
Set_Partner_LU_Name	ok	psc	psc	-	psc	psc
Set_Partner_LU_Name	pc	psc	psc	-	psc	psc
Set_Partner_LU_Name	ps	psc	psc	-	psc	psc
Set_Partner_Port	ok	psc	psc	-	psc	psc
Set_Partner_Port	pc	psc	psc	-	psc	psc
Set_Partner_Port	ps	psc	psc	-	psc	psc
Set_Partner_Tsel	ok	psc	psc	-	psc	psc
Set_Partner_Tsel	pc	psc	psc	-	psc	psc
Set_Partner_Tsel	ps	psc	psc	-	psc	psc
Set_Partner_Tsel_Format	ok	psc	psc	-	psc	psc

Tabelle 19: Zustandstabelle für CPI-C-Aufrufe

Aufruf	**Ergebnis**	**Folgezustand, falls vorher im Zustand**				
		Start	**Reset**	**Init.**	**Send**	**Receive**
Set_Partner_Tsel_Format	pc	psc	psc	-	psc	psc
Set_Partner_Tsel_Format	ps	psc	psc	-	psc	psc
Set_Sync_Level	ok	psc	-	psc	psc	psc
Set_Sync_Level	pc	psc	-	psc	psc	psc
Set_Sync_Level	ps	psc	-	psc	psc	psc
Set_TP_Name	ok	psc	psc	-	psc	psc
Set_TP_Name	pc	psc	psc	-	psc	psc
Specify_Local_Port	ok	psc	-	psc	psc	psc
Specify_Local_Port	pc	psc	-	psc	psc	psc
Specify_Local_Port	ps	psc	-	psc	psc	psc
Specify_Local_Tsel	ok	psc	-	psc	psc	psc
Specify_Local_Tsel	pc	psc	-	psc	psc	psc
Specify_Local_Tsel	ps	psc	-	psc	psc	psc
Specify_Local_Tsel_Format	ok	psc	-	psc	psc	psc
Specify_Local_Tsel_Format	pc	psc	-	psc	psc	psc
Specify_Local_Tsel_Format	ps	psc	-	psc	psc	psc
Specify_Secondary_Return_Code	ok	psc	-	-	-	-
Specify_Secondary_Return_Code	pc	psc	-	-	-	-
Specify_Secondary_Return_Code	ps	psc	-	-	-	-
Enable_UTM_UPIC	ok	Reset	psc	psc	psc	psc
Enable_UTM_UPIC	pc	-	psc	psc	psc	psc
Enable_UTM_UPIC	ps	-	psc	psc	psc	psc
Disable_UTM_UPIC	ok	psc	Start	Start	Start	Start
Disable_UTM_UPIC	pc	psc	-	-	-	-
Disable_UTM_UPIC	ps	psc	-	-	-	-

Tabelle 19: Zustandstabelle für CPI-C-Aufrufe

* Nur unmittelbar nach einem *Receive-*/*Receive_Mapped_Data*-Aufruf erlaubt

Abkürzungen für die Zustandstabelle

Ergebnis	Returncodes
ae	CM_ALLOCATE_FAILURE_RETRY CM_ALLOCATE_FAILURE_NO_RETRY CM_SECURITY_NOT_VALID CM_SECURITY_NOT_SUPPORTED CM_TPN_NOT_RECOGNIZED CM_TP_NOT_AVAILABLE_NO_RETRY CM_TP_NOT_AVAILABLE_RETRY
da	CM_DEALLOCATED_ABEND
dn	CM_DEALLOCATED_NORMAL
oi	CM_OPERATION_INCOMPLETE
ok	CM_OK
pe	CM_PARAMETER_ERROR
pc	CM_PROGRAM_PARAMETER_CHECK
pn	CM_PARAM_VALUE_NOT_SUPPORTED
ps	CM_PRODUCT_SPECIFIC_ERROR
rf	CM_RESOURCE_FAILURE_RETRY CM_RESOURCE_FAILURE_NO_RETRY
nr	CM_NO_SECONDARY_RETURN_CODE
un	CM_OPERATION_UNSUCCESSFUL

Tabelle 20: Abkürzungen für die Zustandstabelle (1)

Ergebnis	data_received and status_received:
dr	CM_COMPLETE_DATA_RECEIVED CM_INCOMPLETE_DATA_RECEIVED
nd	CM_NO_DATA_RECEIVED
no	CM_NO_STATUS_RECEIVED
se	CM_SEND_RECEIVED

Tabelle 21: Abkürzungen für die Zustandstabelle (2)

Folgezustand	Bedeutung
-	keine Zustandsänderung
psc	Fehler CM_PROGRAM_STATE_CHECK

Tabelle 22: Abkürzungen für die Zustandstabelle (3)

Der Returncode CM_CALL_NOT_SUPPORTED ist in der Zustandstabelle nicht enthalten. Er wird zurückgegeben, wenn die UPIC-Bibliothek den Aufruf zwar bereitstellt, die Funktion aber im speziellen Fall nicht unterstützt wird. Es findet keine Zustandsänderung statt.

Fachwörter

Fachwörter, die an anderer Stelle erklärt werden, sind mit *kursiver* Schrift ausgezeichnet.

Ablaufinvariantes Programm
reentrant program

siehe *reentrant-fähiges Programm*.

Abnormale Beendigung einer UTM-Anwendung
abnormal termination of a UTM application

Beendigung einer *UTM-Anwendung*, bei der die *KDCFILE* nicht mehr aktualisiert wird. Eine abnormale Beendigung wird ausgelöst durch einen schwerwiegenden Fehler, z.B. Rechnerausfall, Fehler in der Systemsoftware. Wird die Anwendung erneut gestartet, führt openUTM einen *Warmstart* durch.

abstrakte Syntax (OSI)
abstract syntax

Eine abstrakte Syntax ist die Menge der formal beschriebenen Datentypen, die zwischen Anwendungen über *OSI TP* ausgetauscht werden sollen. Eine abstrakte Syntax ist unabhängig von der eingesetzten Hardware und der jeweiligen Programmiersprache.

Access-List
access list

Eine Access-List definiert die Berechtigung für den Zugriff auf einen bestimmten *Service*, auf eine bestimmte *TAC-Queue* oder auf eine bestimmte *USER-Queue*. Eine Access-List ist als *Keyset* definiert und enthält einen oder mehrere *Keycodes*, die jeweils eine Rolle in der Anwendung repräsentieren. Benutzer, LTERMs oder (OSI-)LPAPs dürfen nur dann auf den Service oder die *TAC-Queue*/*USER-Queue* zugreifen, wenn ihnen die entsprechenden Rollen zugeteilt wurden, d.h. wenn ihr *Keyset* und die Access-List mindestens einen gemeinsamen *Keycode* enthalten.

Access Point (OSI)

siehe *Dienstzugriffspunkt.*

ACID-Eigenschaften
ACID properties
Abkürzende Bezeichnung für die grundlegenden Eigenschaften von *Transaktionen*: Atomicity, Consistency, Isolation und Durability.

Administration
administration
Verwaltung und Steuerung einer *UTM-Anwendung* durch einen *Administrator* oder ein *Administrationsprogramm*.

Administrations-Journal
administration journal
siehe *Cluster-Administrations-Journal*.

Administrationskommando
administration command
Kommandos, mit denen der *Administrator* einer *UTM-Anwendung* Administrationsfunktionen für diese Anwendung durchführt. Die Administrationskommandos sind als *Transaktionscodes* realisiert.

Administrationsprogramm
administration program
Teilprogramm, das Aufrufe der *Programmschnittstelle für die Administration* enthält. Dies kann das Standard-Administrationsprogramm *KDCADM* sein, das mit openUTM ausgeliefert wird, oder ein vom Anwender selbst erstelltes Programm.

Administrator
administrator
Benutzer mit Administrationsberechtigung.

AES
AES (Advanced Encryption Standard) ist der aktuelle symmetrische Verschlüsselungsstandard, festgelegt vom NIST (National Institute of Standards and Technology), basierend auf dem an der Universität Leuven (B) entwickelten Rijndael-Algorithmus. Wird das AES-Verfahren verwendet, dann erzeugt der UPIC-Client für jede Sitzung einen AES-Schlüssel.

Akzeptor (CPI-C)
acceptor
Die Kommunikationspartner einer *Conversation* werden *Initiator* und Akzeptor genannt. Der Akzeptor nimmt die vom Initiator eingeleitete Conversation mit Accept_Conversation entgegen.

Anmelde-Vorgang (KDCS)
sign-on service

Spezieller *Dialog-Vorgang*, bei dem die Anmeldung eines Benutzers an eine UTM-Anwendung durch *Teilprogramme* gesteuert wird.

Anschlussprogramm
linkage program

siehe *KDCROOT*.

Anwendungsinformation
application information

Sie stellt die Gesamtmenge der von der *UTM-Anwendung* benutzten Daten dar. Dabei handelt es sich um Speicherbereiche und Nachrichten der UTM-Anwendung, einschließlich der aktuell auf dem Bildschirm angezeigten Daten. Arbeitet die UTM-Anwendung koordiniert mit einem Datenbanksystem, so gehören die in der Datenbank gespeicherten Daten ebenfalls zur Anwendungsinformation.

Anwendungsprogramm
application program

Ein Anwendungsprogramm bildet den Hauptbestandteil einer *UTM-Anwendung*. Es besteht aus der Main Routine *KDCROOT* und den *Teilprogrammen*. Es bearbeitet alle Aufträge, die an eine *UTM-Anwendung* gerichtet werden.

Anwendungs-Warmstart
application warm start

siehe *Warmstart*.

Apache Axis

Apache Axis (Apache eXtensible Interaction System) ist eine SOAP-Engine zur Konstruktion von darauf basierenden Web Services und Client-Anwendungen. Es existiert eine Implementierung in C++ und Java.

Apache Tomcat

Apache Tomcat stellt eine Umgebung zur Ausführung von Java-Code auf Web-Servern bereit, die im Rahmen des Jakarta-Projekts der Apache Software Foundation entwickelt wird. Es handelt sich um einen in Java geschriebenen Servlet-Container, der mithilfe des JSP-Compilers Jasper auch JavaServer Pages in Servlets übersetzen und ausführen kann. Dazu kommt ein kompletter HTTP-Server.

Application Context (OSI)
application context

Der Application Context ist die Menge der Regeln, die für die Kommunikation zwischen zwei Anwendungen gelten sollen. Dazu gehören z.B. die abstrakten Syntaxen und die zugeordneten *Transfer-Syntaxen*.

Application Entity (OSI)
application entity

Eine Application Entity (AE) repräsentiert alle für die Kommunikation relevanten Aspekte einer realen Anwendung. Eine Application Entity wird durch einen global (d.h. weltweit) eindeutigen Namen identifiziert, den *Application Entity Title* (AET). Jede Application Entity repräsentiert genau einen *Application Process*. Ein Application Process kann mehrere Application Entities umfassen.

Application Entity Title (OSI)
application entity title

Ein Application Entity Title ist ein global (d.h. weltweit) eindeutiger Name für eine *Application Entity*. Er setzt sich zusammen aus dem *Application Process Title* des jeweiligen *Application Process* und dem *Application Entity Qualifier.*

Application Entity Qualifier (OSI)
application entity qualifier

Bestandteil des *Application Entity Titles*. Der Application Entity Qualifier identifiziert einen *Dienstzugriffspunkt* innerhalb der Anwendung. Ein Application Entity Qualifier kann unterschiedlich aufgebaut sein. openUTM unterstützt den Typ „Zahl".

Application Process (OSI)
application process

Der Application Process repräsentiert im *OSI-Referenzmodell* eine Anwendung. Er wird durch den *Application Process Title* global (d.h. weltweit) eindeutig identifiziert.

Application Process Title (OSI)
application process title

Gemäß der OSI-Norm dient der Application Process Title (APT) zur global (d.h. weltweit) eindeutigen Identifizierung von Anwendungen. Er kann unterschiedlich aufgebaut sein. openUTM unterstützt den Typ *Object Identifier*.

Application Service Element (OSI)
application service element

Ein Application Service Element (ASE) repräsentiert eine Funktionsgruppe der Anwendungsschicht (Schicht 7) des *OSI-Referenzmodells*.

Association (OSI)
association

Eine Association ist eine Kommunikationsbeziehung zwischen zwei *Application Entities*. Dem Begriff Association entspricht der *LU6.1*-Begriff *Session*.

Asynchron-Auftrag
queued job

Auftrag, der vom Auftraggeber zeitlich entkoppelt durchgeführt wird. Zur Bearbeitung von Asynchron-Aufträgen sind in openUTM *Message Queuing* Funktionen integriert, vgl. *UTM-gesteuerte Queue* und *Service-gesteuerte Queue*. Ein Asynchron-Auftrag wird durch die *Asynchron-Nachricht*, den Empfänger und ggf. den gewünschten Ausführungszeitpunkt beschrieben.
Ist der Empfänger ein Terminal, ein Drucker oder eine Transportsystem-Anwendung, so ist der Asynchron-Auftrag ein *Ausgabe-Auftrag*; ist der Empfänger ein Asynchron-Vorgang derselben oder einer fernen Anwendung, so handelt es sich um einen *Hintergrund-Auftrag*.
Asynchron-Aufträge können *zeitgesteuerte Aufträge* sein oder auch in einen *Auftrags-Komplex* integriert sein.

Asynchron-Conversation
asynchronous conversation

CPI-C-Conversation, bei der nur der *Initiator* senden darf. Für den *Akzeptor* muss in der *UTM-Anwendung* ein asynchroner Transaktionscode generiert sein.

Asynchron-Nachricht
queued message

Asynchron-Nachrichten sind Nachrichten, die an eine *Message Queue* gerichtet sind. Sie werden von der lokalen *UTM-Anwendung* zunächst zwischengespeichert und dann unabhängig vom Auftraggeber weiter verarbeitet. Je nach Empfänger unterscheidet man folgende Typen von Asynchron-Nachrichten:

- Bei Asynchron-Nachrichten an eine *UTM-gesteuerte Queue* wird die Weiterverarbeitung komplett durch openUTM gesteuert. Zu diesem Typ gehören Nachrichten, die einen lokalen oder fernen *Asynchron-Vorgang* starten (vgl. auch *Hintergrund-Auftrag*) und Nachrichten, die zur Ausgabe an ein Terminal, einen Drucker oder eine Transportsystem-Anwendung geschickt werden (vgl. auch *Ausgabe-Auftrag*).
- Bei Asynchron-Nachrichten an eine *Service-gesteuerte Queue* wird die Weiterverarbeitung durch einen *Service* der Anwendung gesteuert. Zu diesem Typ gehören Nachrichten an eine *TAC-Queue*, Nachrichten an eine *USER-Queue* und Nachrichten an eine *Temporäre Queue*. Die User-Queue und die Temporäre Queue müssen dabei zur lokalen Anwendung gehören, die TAC-Queue kann sowohl in der lokalen als auch in einer fernen Anwendung liegen.

Asynchron-Programm
asynchronous program

Teilprogramm, das von einem *Hintergrund-Auftrag* gestartet wird.

Asynchron-Vorgang (KDCS)
asynchronous service

Vorgang, der einen *Hintergrund-Auftrag* bearbeitet. Die Verarbeitung erfolgt entkoppelt vom Auftraggeber. Ein Asynchron-Vorgang kann aus einem oder mehreren Teilprogrammen/Transaktionen bestehen. Er wird über einen asynchronen *Transaktionscode* gestartet.

Auftrag
job

Anforderung eines *Services*, der von einer *UTM-Anwendung* zur Verfügung gestellt wird, durch Angabe eines Transaktionscodes. Siehe auch: *Ausgabe-Auftrag*, *Dialog-Auftrag*, *Hintergrund-Auftrag*, *Auftrags-Komplex*.

Auftraggeber-Vorgang
job-submitting service

Ein Auftraggeber-Vorgang ist ein *Vorgang*, der zur Bearbeitung eines Auftrags einen Service von einer anderen Server-Anwendung (*Auftragnehmer-Vorgang*) anfordert.

Auftragnehmer-Vorgang
job-receiving service

Ein Auftragnehmer-Vorgang ist ein *Vorgang*, der von einem *Auftraggeber-Vorgang* einer anderen Server-Anwendung gestartet wird.

Auftrags-Komplex
job complex

Auftrags-Komplexe dienen dazu, *Asynchron-Aufträgen Quittungsaufträge* zuzuordnen. Ein Asynchron-Auftrag innerhalb eines Auftrags-Komplexes wird *Basis-Auftrag* genannt.

Ausgabe-Auftrag
queued output job

Ausgabeaufträge sind *Asynchron-Aufträge*, die die Aufgabe haben, eine Nachricht, z.B. ein Dokument, an einen Drucker ein Terminal oder eine Transportsystem-Anwendung auszugeben.
Ausgabeaufträge werden ausschließlich von UTM-Systemfunktionen bearbeitet, d.h. für die Bearbeitung müssen keine Teilprogramme erstellt werden.

Authentisierung
authentication
siehe *Zugangskontrolle*.

Autorisierung
authorization
siehe *Zugriffskontrolle*.

Axis
siehe *Apache Axis*.

Basis-Auftrag
basic job
Asynchron-Auftrag in einem *Auftrags-Komplex*.

Basisformat
basic format
Format, in das der Terminal-Benutzer alle Angaben eintragen kann, die notwendig sind, um einen Vorgang zu starten.

Benutzerausgang
user exit
Begriff ersetzt durch *Event-Exit.*

Benutzerkennung
user ID
Bezeichner für einen Benutzer, der in der *Konfiguration* der *UTM-Anwendung* festgelegt ist (optional mit Passwort zur *Zugangskontrolle*) und dem spezielle Zugriffsrechte (*Zugriffskontrolle*) zugeordnet sind. Ein Terminal-Benutzer muss bei der Anmeldung an die UTM-Anwendung diesen Bezeichner (und ggf. das zugeordnete Passwort) angeben.
Für andere Clients ist die Angabe der Benutzerkennung optional, siehe auch *Verbindungs-Benutzerkennung*.
UTM-Anwendungen können auch ohne Benutzerkennungen generiert werden.

Benutzer-Protokolldatei
user log file
Datei oder Dateigeneration, in die der Benutzer mit dem KDCS-Aufruf LPUT Sätze variabler Länge schreibt. Jedem Satz werden die Daten aus dem KB-Kopf des *KDCS-Kommunikationsbereichs* vorangestellt. Die Benutzerprotokolldatei unterliegt der Transaktionssicherung von openUTM.

Berechtigungsprüfung
sign-on check

siehe *Zugangskontrolle*.

Beweissicherung (BS2000/OSD)
audit

Im Betrieb einer *UTM-Anwendung* werden zur Beweissicherung sicherheitsrelevante UTM-Ereignisse von *SAT* protokolliert.

Bildschirm-Wiederanlauf
screen restart

Wird ein *Dialog-Vorgang* unterbrochen, gibt openUTM beim *Vorgangswiederanlauf* die *Dialog-Nachricht* der letzten abgeschlossenen *Transaktion* erneut auf dem Bildschirm aus, sofern die letzte Transaktion eine Nachricht auf den Bildschirm ausgegeben hat.

Browsen von Asynchron-Nachrichten
browsing asynchronous messages

Ein *Vorgang* liest nacheinander die *Asynchron-Nachrichten*, die sich in einer *Service-gesteuerten Queue* befinden. Die Nachrichten werden während des Lesens nicht gesperrt und verbleiben nach dem Lesen in der Queue. Dadurch ist gleichzeitiges Lesen durch unterschiedliche Vorgänge möglich.

Bypass-Betrieb (BS2000/OSD)
bypass mode

Betriebsart eines Druckers, der lokal an ein Terminal angeschlossen ist. Im Bypass-Betrieb wird eine an den Drucker gerichtete *Asynchron-Nachricht* an das Terminal gesendet und von diesem auf den Drucker umgeleitet, ohne auf dem Bildschirm angezeigt zu werden.

Cache-Speicher
cache

Pufferbereich zur Zwischenspeicherung von Anwenderdaten für alle Prozesse einer *UTM-Anwendung*. Der Cache-Speicher dient zur Optimierung der Zugriffe auf den *Pagepool* und für UTM-Cluster-Anwendungen zusätzlich auf den *Cluster-Pagepool*.

CCS-Name (BS2000/OSD)
CCS name

siehe *Coded-Character-Set-Name*.

Client
client

Clients einer *UTM-Anwendung* können sein:
- Terminals
- UPIC-Client-Programme
- Transportsystem-Anwendungen (z.B. DCAM-, PDN-, CMX-, Socket-Anwendungen oder UTM-Anwendungen, die als *Transportsystem-Anwendung* generiert sind)

Clients werden über LTERM-Partner an die UTM-Anwendung angeschlossen. openUTM-Clients mit Trägersystem OpenCPIC werden wie *OSI TP-Partner* behandelt.

Client-Seite einer Conversation
client side of a conversation

Begriff ersetzt durch *Initiator*.

Cluster

Eine Anzahl von Rechnern, die über ein schnelles Netzwerk verbunden sind und die von außen in vielen Fällen als ein Rechner gesehen werden können. Das Ziel des „Clustering“ ist meist die Erhöhung der Rechenkapazität oder der Verfügbarkeit gegenüber einem einzelnen Rechner.

Cluster-Administrations-Journal
cluster administration journal

Die Administrations-Journal-Dateien dienen dazu, administrative Aktionen, die in einer UTM-Cluster-Anwendung Cluster-weit auf alle Knoten-Anwendungen wirken sollen, an die anderen Knoten-Anwendungen weiterzugeben.

Cluster-GSSB-Datei
cluster GSSB file

Datei zur Verwaltung von GSSBs in einer *UTM-Cluster-Anwendung*. Die Cluster-GSSB-Datei wird mit dem UTM-Generierungstool *KDCDEF* erstellt.

Cluster-Konfigurationsdatei
cluster configuration file

Datei, die die zentralen Konfigurationsdaten einer *UTM-Cluster-Anwendung* enthält. Die Cluster-Konfigurationsdatei wird mit dem UTM-Generierungstool *KDCDEF* erstellt.

Cluster-Lock-Datei
cluster lock file

Datei einer *UTM-Cluster-Anwendung*, die dazu dient, Knoten-übergreifende Sperren auf Anwenderdatenbereiche zu verwalten.

Cluster-Pagepool
cluster pagepool

Der Cluster-Pagepool besteht aus einer Verwaltungsdatei und bis zu 10 Dateien, in denen die Cluster-weit verfügbaren Anwenderdaten (Vorgangsdaten inklusive LSSB, GSSB und ULS) einer *UTM-Cluster-Anwendung* gespeichert werden. Der Cluster-Pagepool wird mit dem UTM-Generierungstool *KDCDEF* erstellt.

Cluster-Startserialisierungs-Datei
cluster start serialization file

Lock-Datei, mit der die Starts einzelner Knoten-Anwendungen serialisiert werden (nur bei Unix- und Windows-Systemen).

Cluster-ULS-Datei
cluster ULS file

Datei zur Verwaltung von ULS-Bereichen einer *UTM-Cluster-Anwendung*. Die Cluster-ULS-Datei wird mit dem UTM-Generierungstool *KDCDEF* erstellt.

Cluster-User-Datei
cluster user file

Datei, die die Verwaltungsdaten der Benutzer einer *UTM-Cluster-Anwendung* enthält. Die Cluster-User-Datei wird mit dem UTM-Generierungstool *KDCDEF* erstellt.

Coded-Character-Set-Name (BS2000/OSD)
coded character set name

Bei Verwendung des Produkts *XHCS* (e**X**tended **H**ost **C**ode **S**upport) wird jeder verwendete Zeichensatz durch einen Coded-Character-Set-Namen (abgekürzt: „CCS-Name“ oder „CCSN“) eindeutig identifiziert.

Communication Resource Manager
communication resource manager

Communication Resource Manager (CRMs) kontrollieren in verteilten Systemen die Kommunikation zwischen den Anwendungsprogrammen. openUTM stellt CRMs für den internationalen Standard OSI TP, für den Industrie-Standard *LU6.1* und für das openUTM-eigene Protokoll UPIC zur Verfügung.

Contention Loser
contention loser

Jede Verbindung zwischen zwei Partnern wird von einem der Partner verwaltet. Der Partner, der die Verbindung verwaltet, heißt *Contention Winner*. Der andere Partner ist der Contention Loser.

Contention Winner
contention winner
Der Contention Winner einer Verbindung übernimmt die Verwaltung der Verbindung. Aufträge können sowohl vom Contention Winner als auch vom *Contention Loser* gestartet werden. Im Konfliktfall, wenn beide Kommunikationspartner gleichzeitig einen Auftrag starten wollen, wird die Verbindung vom Auftrag des Contention Winner belegt.

Conversation
conversation
Bei CPI-C nennt man die Kommunikation zwischen zwei CPI-C-Anwendungsprogrammen Conversation. Die Kommunikationspartner einer Conversation werden *Initiator* und *Akzeptor* genannt.

Conversation-ID
conversation ID
Jeder *Conversation* wird von CPI-C lokal eine Conversation-ID zugeordnet, d.h. *Initiator* und *Akzeptor* haben jeweils eine eigene Conversation-ID. Mit der Conversation-ID wird jeder CPI-C-Aufruf innerhalb eines Programms eindeutig einer Conversation zugeordnet.

CPI-C
CPI-C (Common Programming Interface for Communication) ist eine von X/Open und dem CIW (**C**PI-C **I**mplementor's **W**orkshop) normierte Programmschnittstelle für die Programm-Programm-Kommunikation in offenen Netzen. Das in openUTM implementierte CPI-C genügt der CPI-C V2.0 CAE Specification von X/Open. Die Schnittstelle steht in COBOL und C zur Verfügung. CPI-C in openUTM kann über die Protokolle OSI TP, LU6.1, UPIC und mit openUTM-LU6.2 kommunizieren.

Cross Coupled System / XCS
Verbund von BS2000-Rechnern mit *Highly Integrated System Complex* Multiple System Control Facility (HIPLEX® MSCF).

Dead Letter Queue
dead letter queue
Die Dead Letter Queue ist eine TAC-Queue mit dem festen Namen KDCDLETQ. Sie steht immer zur Verfügung, um Asynchron-Nachrichten an Transaktionscodes oder TAC-Queues zu sichern, die nicht verarbeitet werden konnten. Die Sicherung von Asynchron-Nachrichten in der Dead Letter Queue kann durch den Parameter DEAD-LETTER-Q der TAC-Anweisung für jedes Nachrichtenziel einzeln ein- und ausgeschaltet werden.

DES

DES (Data Encryption Standard) ist eine internationale Norm zur Verschlüsselung von Daten. Bei diesem Verfahren wird ein Schlüssel zum Ver- und Entschlüsseln verwendet. Wird das DES-Verfahren verwendet, dann erzeugt der UPIC-Client für jede Sitzung einen DES-Schlüssel.

Dialog-Auftrag
dialog job, interactive job

Auftrag, der einen *Dialog-Vorgang* startet. Der Auftrag kann von einem *Client* oder - bei *Server-Server-Kommunikation* - von einer anderen Anwendung erteilt werden.

Dialog-Conversation
dialog conversation

CPI-C-Conversation, bei der sowohl der *Initiator* als auch der *Akzeptor* senden darf. Für den *Akzeptor* muss in der *UTM-Anwendung* ein Dialog-Transaktionscode generiert sein.

Dialog-Nachricht
dialog message

Nachricht, die eine Antwort erfordert oder selbst eine Antwort auf eine Anfrage ist. Dabei bilden Anfrage und Antwort einen *Dialog-Schritt*.

Dialog-Programm
dialog program

Teilprogramm, das einen *Dialog-Schritt* teilweise oder vollständig bearbeitet.

Dialog-Schritt
dialog step

Ein Dialog-Schritt beginnt mit dem Empfang einer *Dialog-Nachricht* durch die *UTM-Anwendung*. Er endet mit der Antwort der UTM-Anwendung.

Dialog-Terminalprozess (Unix-/Windows-Systeme)
dialog terminal process

Ein Dialog-Terminalprozess verbindet ein Unix-/Windows-Terminal mit den *Workprozessen* der *UTM-Anwendung*. Dialog-Terminalprozesse werden entweder vom Benutzer durch Eingabe von utmdtp oder über die LOGIN-Shell gestartet. Für jedes Terminal, das an eine UTM-Anwendung angeschlossen werden soll, ist ein eigener Dialog-Terminalprozess erforderlich.

Dialog-Vorgang
dialog service

Vorgang, der einen *Auftrag* im Dialog (zeitlich gekoppelt) mit dem Auftraggeber (*Client* oder eine andere Server-Anwendung) bearbeitet. Ein Dialog-Vorgang verarbeitet *Dialog-Nachrichten* vom Auftraggeber und erzeugt Dialog-Nachrichten für diesen. Ein Dialog-Vorgang besteht aus mindestens einer *Transaktion.* Ein Dialog-Vorgang umfasst in der Regel mindestens einen *Dialog-Schritt*. Ausnahme: Bei *Vorgangskettung* können auch mehrere Vorgänge einen Dialog-Schritt bilden.

Dienst
service

Programm auf Windows-Systemen, das im Hintergrund unabhängig von angemeldeten Benutzern oder Fenstern abläuft.

Dienstzugriffspunkt
service access point

Im *OSI-Referenzmodell* stehen einer Schicht am Dienstzugriffspunkt die Leistungen der darunterliegenden Schicht zur Verfügung. Der Dienstzugriffspunkt wird im lokalen System durch einen *Selektor* identifiziert. Bei der Kommunikation bindet sich die *UTM-Anwendung* an einen Dienstzugriffspunkt. Eine Verbindung wird zwischen zwei Dienstzugriffspunkten aufgebaut.

Distributed Lock Manager / DLM (BS2000/OSD)

Konkurrierende, Rechner-übergreifende Dateizugriffe können über den Distributed Lock Manager synchronisiert werden.
DLM ist eine Basisfunktion von HIPLEX® MSCF.

Distributed Transaction Processing

X/Open-Architekturmodell für die transaktionsorientierte *verteilte Verarbeitung*.

Druckadministration
print administration

Funktionen zur *Drucksteuerung* und Administration von *Ausgabeaufträgen*, die an einen Drucker gerichtet sind.

Druckerbündel
printer pool

Mehrere Drucker, die demselben *LTERM-Partner* zugeordnet sind.

Druckergruppe (Unix-Systeme)
printer group

Die Unix-Plattform richtet für jeden Drucker standardmäßig eine Druckergruppe ein, die genau diesen Drucker enthält. Darüber hinaus lassen sich mehrere Drucker einer Druckergruppe, aber auch ein Drucker mehreren Druckergruppen zuordnen.

Druckerprozess (Unix-Systeme)
printer process

Prozess, der vom *Mainprozess* zur Ausgabe von *Asynchron-Nachrichten* an eine *Druckergruppe* eingerichtet wird. Er existiert, solange die Druckergruppe an die *UTM-Anwendung* angeschlossen ist. Pro angeschlossener Druckergruppe gibt es einen Druckerprozess.

Druckersteuerstation
printer control terminal

Begriff wurde ersetzt durch *Druckersteuer-LTERM*.

Druckersteuer-LTERM
printer control LTERM

Über ein Druckersteuer-LTERM kann sich ein *Client* oder ein Terminal-Benutzer an eine *UTM-Anwendung* anschließen. Von dem Client-Programm oder Terminal aus kann dann die *Administration* der Drucker erfolgen, die dem Druckersteuer-LTERM zugeordnet sind. Hierfür ist keine Administrationsberechtigung notwendig.

Drucksteuerung
print control

openUTM-Funktionen zur Steuerung von Druckausgaben.

Dynamische Konfiguration
dynamic configuration

Änderung der *Konfiguration* durch die Administration. Im laufenden Betrieb der Anwendung können UTM-Objekte wie z.B. *Teilprogramme*, *Transaktionscodes*, *Clients*, *LU6.1-Verbindungen*, Drucker oder *Benutzerkennungen* in die Konfiguration aufgenommen, modifiziert oder teilweise auch gelöscht werden. Hierzu kann das Administrationsprogramm WinAdmin verwendet werden, oder es müssen eigene *Administrationsprogramme* erstellt werden, die die Funktionen der *Programmschnittstelle der Administration* nutzen.

Einschritt-Transaktion
single-step transaction

Transaktion, die genau einen *Dialog-Schritt* umfasst.

Einschritt-Vorgang
single-step service
Dialog-Vorgang, der genau einen *Dialog-Schritt* umfasst.

Ereignisgesteuerter Vorgang
event-driven service
Begriff ersetzt durch *Event-Service*.

Event-Exit
event exit
Routine des *Anwendungsprogramms*, das bei bestimmten Ereignissen (z.B. Start eines Prozesses, Ende eines Vorgangs) automatisch gestartet wird. Diese darf - im Gegensatz zu den *Event-Services* - keine KDCS-, CPI-C- und XATMI-Aufrufe enthalten.

Event-Funktion
event function
Oberbegriff für *Event-Exits* und *Event-Services*.

Event-Service
event service
Vorgang, der beim Auftreten bestimmter Ereignisse gestartet wird, z.B. bei bestimmten UTM-Meldungen. Die *Teilprogramme* ereignisgesteuerter Vorgänge müssen KDCS-Aufrufe enthalten.

Generierung
generation
Statische Konfiguration einer *UTM-Anwendung* mit dem UTM-Tool KDCDEF und Erzeugen des Anwendungsprogramms.

Globaler Sekundärer Speicherbereich/GSSB
global secondary storage area
siehe *Sekundärspeicherbereich.*

Hardcopy-Betrieb
hardcopy mode
Betriebsart eines Druckers, der lokal an ein Terminal angeschlossen ist. Dabei wird eine Nachricht, die auf dem Bildschirm angezeigt wird, zusätzlich auf dem Drucker abgedruckt.

Heterogene Kopplung
heterogeneous link
Bei *Server-Server-Kommunikation:* Kopplung einer *UTM-Anwendung* mit einer Nicht-UTM-Anwendung, z.B. einer CICS- oder TUXEDO-Anwendung.

Highly Integrated System Complex / HIPLEX®

Produktfamilie zur Realisierung eines Bedien-, Last- und Verfügbarkeitsverbunds mit mehreren BS2000-Servern.

Hintergrund-Auftrag
background job

Hintergrund-Aufträge sind *Asynchron-Aufträge*, die an einen *Asynchron-Vorgang* der eigenen oder einer fernen Anwendung gerichtet sind. Hintergrund-Aufträge eignen sich besonders für zeitintensive oder zeitunkritische Verarbeitungen, deren Ergebnis keinen direkten Einfluss auf den aktuellen Dialog hat.

HIPLEX® MSCF

(MSCF = **M**ultiple **S**ystem **C**ontrol **F**acility)
stellt bei HIPLEX® die Infrastruktur sowie Basisfunktionen für verteilte Anwendungen bereit.

Homogene Kopplung
homogeneous link

Bei *Server-Server-Kommunikation*: Kopplung von *UTM-Anwendungen.* Dabei spielt es keine Rolle, ob die Anwendungen auf der gleichen oder auf unterschiedlichen Betriebssystem-Plattformen ablaufen.

Inbound-Conversation (CPI-C)
inbound conversation

siehe *Incoming-Conversation.*

Incoming-Conversation (CPI-C)
incoming conversation

Eine *Conversation*, bei der das lokale CPI-C-Programm *Akzeptor* ist, heißt Incoming-Conversation. In der X/Open-Specification wird für Incoming-Conversation auch das Synonym Inbound-Conversation verwendet.

Initiale KDCFILE
initial KDCFILE

In einer *UTM-Cluster-Anwendung* die *KDCFILE*, die von *KDCDEF* erzeugt wurde und vor dem Start der Knoten-Anwendungen für jeden Knoten kopiert werden muss.

Initiator (CPI-C)
initiator

Die Kommunikationspartner einer *Conversation* werden Initiator und *Akzeptor* genannt. Der Initiator baut die Conversation mit den CPI-C-Aufrufen Initialize_Conversation und Allocate auf.

Insert
insert

Feld in einem Meldungstext, in das openUTM aktuelle Werte einträgt.

Inverser KDCDEF
inverse KDCDEF

Funktion, die aus den Konfigurationsdaten der *KDCFILE*, die im laufenden Betrieb dynamisch angepasst wurde, Steueranweisungen für einen *KDCDEF*-Lauf erzeugt. Der inverse KDCDEF kann „offline“ unter KDCDEF oder „online“ über die *Programmschnittstelle zur Administration* gestartet werden.

JDK

Java Development Kit
Standard-Entwicklungsumgebung von Sun Microsystems für die Entwicklung von Java-Anwendungen.

Kaltstart
cold start

Starten einer *UTM-Anwendung* nach einer *normalen Beendigung* der Anwendung oder nach einer Neugenerierung (vgl. auch *Warmstart*).

KDCADM

Standard-Administrationsprogramm, das zusammen mit openUTM ausgeliefert wird. KDCADM stellt Administrationsfunktionen zur Verfügung, die über Transaktionscodes (*Administrationskommandos*) aufgerufen werden.

KDCDEF

UTM-Tool für die *Generierung* von *UTM-Anwendungen*. KDCDEF erstellt anhand der Konfigurationsinformationen in den KDCDEF-Steueranweisungen die UTM-Objekte *KDCFILE* und die ROOT-Tabellen-Source für die Main Routine *KDCROOT*.
In UTM-Cluster-Anwendungen erstellt KDCDEF zusätzlich die *Cluster-Konfigurationsdatei,* die *Cluster-User-Datei*, den *Cluster-Pagepool*, die *Cluster-GSSB-Datei* und die *Cluster-ULS-Datei*.

KDCFILE

Eine oder mehrere Dateien, die für den Ablauf einer *UTM-Anwendung* notwendige Daten enthalten. Die KDCFILE wird mit dem UTM-Generierungstool *KDCDEF* erstellt. Die KDCFILE enthält unter anderem die *Konfiguration* der Anwendung.

KDCROOT

Main Routine eines *Anwendungsprogramms*, die das Bindeglied zwischen *Teilprogrammen* und UTM-Systemcode bildet. KDCROOT wird zusammen mit den *Teilprogrammen* zum *Anwendungsprogramm* gebunden.

KDCS-Parameterbereich
KDCS parameter area

siehe *Parameterbereich.*

KDCS-Programmschnittstelle
KDCS program interface

Universelle UTM-Programmschnittstelle, die den nationalen Standard DIN 66 265 erfüllt und Erweiterungen enthält. Mit KDCS (Kompatible Datenkommunikationsschnittstelle) lassen sich z.B. Dialog-Services erstellen und *Message Queuing* Funktionen nutzen. Außerdem stellt KDCS Aufrufe zur *verteilten Verarbeitung* zur Verfügung.

Kerberos

Kerberos ist ein standardisiertes Netzwerk-Authentisierungsprotokoll (RFC1510), das auf kryptographischen Verschlüsselungsverfahren basiert, wobei keine Kennwörter im Klartext über das Netzwerk gesendet werden.

Kerberos-Principal
Kerberos principal

Eigentümer eines Schlüssels.
Kerberos arbeitet mit symmetrischer Verschlüsselung, d.h. alle Schlüssel liegen an zwei Stellen vor, beim Eigentümer eines Schlüssels (Principal) und beim KDC (Key Distribution Center).

Keycode
key code

Code, der in einer Anwendung eine bestimmte Zugriffsberechtigung oder eine bestimmte Rolle repräsentiert. Mehrere Keycodes werden zu einem *Keyset* zusammengefasst.

Keyset
key set

Zusammenfassung von einem oder mehrerer *Keycodes* unter einem bestimmten Namen. Ein Keyset definiert Berechtigungen im Rahmen des verwendeten Berechtigungskonzepts (Lock-/Keycode-Konzept oder *Access-List*-Konzept). Ein Keyset kann einer *Benutzerkennung*, einem *LTERM-Partner*, einem (OSI-)*LPAP-Partner*, einem *Service* oder einer *TAC-Queue* zugeordnet werden.

Knoten
node

Einzelner Rechner eines *Clusters*.

Knoten-Anwendung
node application

UTM-Anwendung, die als Teil einer *UTM-Cluster-Anwendung* auf einem einzelnen *Knoten* zum Ablauf kommt.

Knotengebundener Vorgang
node bound service

Ein knotengebundener Vorgang eines Benutzers kann nur an dem Knoten fortgesetzt werden, an dem der Benutzer zuletzt angemeldet war. Folgende Vorgänge sind immer knotengebunden:

- Vorgänge, die eine Kommunikation mit einem Auftragnehmer über LU6.1 oder OSI TP begonnen haben und bei denen der Auftragnehmervorgang noch nicht beendet wurde
- eingeschobene Vorgänge einer Vorgangskellerung
- Vorgänge, die eine SESAM-Transaktion abgeschlossen haben

Außerdem ist der Vorgang eines Benutzers knotengebunden, solange der Benutzer an eine Knoten-Anwendung angemeldet ist.

Kommunikationsbereich/KB (KDCS)
communication area

Transaktionsgesicherter KDCS-*Primärspeicherbereich*, der Vorgangs-spezifische Daten enthält. Der Kommunikationsbereich besteht aus 3 Teilen:

- dem KB-Kopf mit allgemeinen Vorgangsdaten,
- dem KB-Rückgabebereich für Rückgaben nach KDCS-Aufrufen
- dem KB-Programmbereich zur Datenübergabe zwischen UTM-Teilprogrammen innerhalb eines *Vorgangs*.

Konfiguration
configuration

Summe aller Eigenschaften einer *UTM-Anwendung*. Die Konfiguration beschreibt:

- Anwendungs- und Betriebsparameter
- die Objekte der Anwendung und die Eigenschaften dieser Objekte. Objekte sind z.B. *Teilprogramme* und *Transaktionscodes*, Kommunikationspartner, Drucker, *Benutzerkennungen*
- definierte Zugriffsschutz- und Zugangsschutzmaßnahmen

Die Konfiguration einer UTM-Anwendung wird bei der Generierung festgelegt und kann per *Administration* dynamisch (während des Anwendungslaufs) geändert werden. Die Konfiguration ist in der *KDCFILE* abgelegt.

Konfiguration
configuration

Festlegen der *Konfiguration* der UTM-Anwendung. Es wird unterschieden zwischen *statischer* und *dynamischer Konfiguration*.

Logische Verbindung
virtual connection

Zuordnung zweier Kommunikationspartner.

Log4j

Log4j ist ein Teil des Apache Jakarta Projekts. Log4j bietet Schnittstellen zum Protokollieren von Informationen (Ablauf-Informationen, Trace-Records,...) und zum Konfigurieren der Protokoll-Ausgabe. *WS4UTM* verwendet das Softwareprodukt Log4j für die Trace- und Logging-Funktionalität.

Lockcode

Code, um einen LTERM-Partner oder einen Transaktionscode vor unberechtigtem Zugriff zu schützen. Damit ist ein Zugriff nur möglich, wenn das *Keyset* des Zugreifenden den passenden *Keycode* enthält (Lock-/Keycode-Konzept).

Lokaler Sekundärer Speicherbereich/LSSB
local secondary storage area

siehe *Sekundärspeicherbereich.*

LPAP-Bündel
LPAP bundle

LPAP-Bündel ermöglichen die Verteilung von Nachrichten an LPAP-Partner auf mehrere Partner-Anwendungen. Soll eine UTM-Anwendung sehr viele Nachrichten mit einer Partner-Anwendung austauschen, kann es für die Lastverteilung sinnvoll sein, mehrere Instanzen der Partner-Anwendung zu starten und die Nachrichten auf die einzelnen Instanzen zu verteilen. In einem LPAP-Bündel übernimmt *open*UTM die Verteilung der Nachrichten an die Instanzen der Partner-Anwendung. Ein LPAP-Bündel besteht aus einem Master-LPAP und mehreren Slave-LPAPs. Die Slave-LPAPs werden dem Master-LPAP bei der Generierung zugeordnet. LPAP-Bündel gibt es sowohl für das OSI TP-Protokoll als auch für das LU6.1-Protokoll.

LPAP-Partner
LPAP partner

Für die *verteilte Verarbeitung* über das *LU6.1*-Protokoll muss in der lokalen Anwendung für jede Partneranwendung ein LPAP-Partner konfiguriert werden. Der LPAP-Partner spiegelt in der lokalen Anwendung die Partneranwendung wider. Bei der Kommunikation wird die Partneranwendung nicht über ihren Anwendungsnamen oder ihre Adresse, sondern über den Namen des zugeordneten LPAP-Partners angesprochen.

LTERM-Bündel
LTERM bundle

Ein LTERM-Bündel (Verbindungsbündel) besteht aus einem Master-LTERM und mehreren Slave-LTERMs. Mit einem LTERM-Bündel (Verbindungsbündel) verteilen Sie asynchrone Nachrichten an eine logische Partner-Anwendung gleichmäßig auf mehrere parallele Verbindungen.

LTERM-Gruppe
LTERM group

Eine LTERM-Gruppe besteht aus einem oder mehreren Alias-LTERMs, den Gruppen-LTERMs, und einem Primary-LTERM. In einer LTERM-Gruppe ordnen Sie mehrere LTERMs einer Verbindung zu.

LTERM-Partner
LTERM partner

Um *Clients* oder Drucker an eine *UTM-Anwendung* anschließen zu können, müssen in der Anwendung LTERM-Partner konfiguriert werden. Ein Client oder Drucker kann nur angeschlossen werden, wenn ihm ein LTERM-Partner mit entsprechenden Eigenschaften zugeordnet ist. Diese Zuordnung wird i.A. bei der Konfiguration festgelegt, sie kann aber auch dynamisch über Terminal-Pools erfolgen.

LTERM-Pool
LTERM pool

Statt für jeden *Client* eine LTERM- und eine PTERM-Anweisung anzugeben, kann mit der Anweisung TPOOL ein Pool von LTERM-Partnern definiert werden. Schließt sich ein Client über einen LTERM-Pool an, wird ihm dynamisch ein LTERM-Partner aus dem Pool zugeordnet.

LU6.1

Geräteunabhängiges Datenaustauschprotokoll (Industrie-Standard) für die transaktionsgesicherte *Server-Server-Kommunikation*.

LU6.1-LPAP-Bündel
LU6.1-LPAP bundle

LPAP-Bündel für *LU6.1*-Partner-Anwendungen.

Mainprozess (Unix-/Windows-Systeme)
main process

Prozess, der die *UTM-Anwendung* startet. Er startet die *Workprozesse*, die *Druckerprozesse, Netzprozesse* und den *Timerprozess* und überwacht die *UTM-Anwendung*.

Main Routine KDCROOT
main routine KDCROOT

siehe *KDCROOT*.

Mapped Hostname
mapped host name

Abbildung des UTM-Hostnamen der Partner-Anwendung in einen realen Hostnamen oder umgekehrt.

Meldung / UTM-Meldung
UTM message

Meldungen werden vom Transaktionsmonitor openUTM oder von UTM-Tools (wie z.B. *KDCDEF*) an *Meldungsziele* ausgegeben. Eine Meldung besteht aus einer Meldungsnummer und dem Meldungstext, der ggf. *Inserts* mit aktuellen Werten enthält. Je nach Meldungsziel werden entweder die gesamte Meldung oder nur Teile der Meldung (z.B. nur die Inserts) ausgegeben.

Meldungsdefinitionsdatei
message definition file

Die Meldungsdefinitionsdatei wird mit openUTM ausgeliefert und enthält standardmäßig die UTM-Meldungstexte in deutscher und englischer Sprache und die Definitionen der Meldungseigenschaften. Aufbauend auf diese Datei kann der Anwender auch eigene, individuelle Meldungsmodule erzeugen.

Meldungsziel
message destination

Ausgabemedium für eine *Meldung*. Mögliche Meldungsziele von Meldungen des Transaktionsmonitors openUTM sind z.B. Terminals, *TS-Anwendungen*, der *Event-Service* MSGTAC, die *System-Protokolldatei* SYSLOG oder *TAC-Queues, Asynchron-TACs, USER-Queues*, SYSOUT/SYSLST bzw. stderr/stdout.
Meldungsziele von Meldungen der UTM-Tools sind SYSOUT/SYSLST bzw. stderr/stdout.

Mehrschritt-Transaktion
multi-step transaction
Transaktion, die aus mehr als einem *Verarbeitungsschritt* besteht.

Mehrschritt-Vorgang (KDCS)
multi-step service
Vorgang, der in mehreren *Dialog-Schritten* ausgeführt wird.

Message Queuing
message queuing
Message Queuing (MQ) ist eine Form der Kommunikation, bei der die Nachrichten (Messages) nicht unmittelbar, sondern über zwischengeschaltete *Message Queues* ausgetauscht werden. Sender und Empfänger können zeitlich und räumlich entkoppelt ablaufen, die Übermittlung der Nachricht wird garantiert, unabhängig davon, ob gerade eine Netzverbindung besteht oder nicht. Bei openUTM gibt es *UTM-gesteuerte Queues* und *Service-gesteuerte Queues*.

Message Queue
message queue
Warteschlange, in der bestimmte Nachrichten transaktionsgesichert bis zur Weiterverarbeitung eingereiht werden. Je nachdem, wer die Weiterverarbeitung kontrolliert, unterscheidet man *Service-gesteuerte Queues* und *UTM-gesteuerte Queues*.

MSGTAC
MSGTAC
Spezieller Event-Service, der Meldungen mit dem Meldungsziel MSGTAC per Programm verarbeitet. MSGTAC ist ein Asynchron-Vorgang und wird vom Betreiber der Anwendung erstellt.

Multiplexanschluss (BS2000/OSD)
multiplex connection
Spezielle Möglichkeit, Terminals an eine *UTM-Anwendung* anzuschließen. Ein Multiplexanschluss ermöglicht es, dass sich mehrere Terminals eine *Transportverbindung* teilen.

Nachrichten-Bereich/NB (KDCS)
KDCS message area
Bei KDCS-Aufrufen: Puffer-Bereich, in dem Nachrichten oder Daten für openUTM oder für das *Teilprogramm* bereitgestellt werden.

Nachrichten-Verteiler (BS2000/OSD)
message router
Einrichtung in einem zentralen Rechner oder Kommunikationsrechner zur Verteilung von Eingabe-Nachrichten an unterschiedliche *UTM-Anwendungen*, die auf unterschiedlichen Rechnern liegen können. Der Nachrichten-Verteiler ermöglicht außerdem, mit *Multiplexanschlüssen* zu arbeiten.

Network File System/Service / NFS
Ermöglicht den Zugriff von Unix-Rechnern auf Dateisysteme über das Netzwerk.

Netzprozess (Unix-/Windows-Systeme)
net process
Prozess einer *UTM-Anwendung* zur Netzanbindung.

Netzwerk-Selektor
network selector
Der Netzwerk-Selektor identifiziert im lokalen System einen *Dienstzugriffspunkt* zur Vermittlungsschicht des *OSI-Referenzmodells*.

Normale Beendigung einer UTM-Anwendung
normal termination of a UTM application
Kontrollierte Beendigung einer *UTM-Anwendung*; das bedeutet u.a., dass die Verwaltungsdaten auf der *KDCFILE* aktualisiert werden. Eine normale Beendigung veranlasst der *Administrator* (z.B. mit KDCSHUT N). Den Start nach einer normalen Beendigung führt openUTM als *Kaltstart* durch.

Object Identifier
object identifier
Ein Object Identifier ist ein global (d.h. weltweit) eindeutiger Bezeichner für Objekte im OSI-Umfeld. Ein Object Identifier besteht aus einer Folge von ganzen Zahlen, die einen Pfad in einer Baumstruktur repräsentiert.

Offener Terminalpool
open terminal pool
Terminalpool, der nicht auf *Clients* eines Rechners oder eines bestimmten Typs beschränkt ist. An diesen Terminalpool können sich alle Clients anschließen, für die kein Rechner- oder Typ-spezifischer Terminalpool generiert ist.

Online-Import
online import
Als Online-Import wird in einer *UTM-Cluster-Anwendung* das Importieren von Anwendungsdaten aus einer normal beendeten Knoten-Anwendung in eine laufende Knoten-Anwendung bezeichnet.

Online-Update
online update

Als Online-Update wird in einer *UTM-Cluster-Anwendung* die Änderung der Konfiguration der Anwendung oder des Anwendungsprogramms oder der Einsatz einer neuen UTM-Korrekturstufe bei laufender *UTM-Cluster-Anwendung* bezeichnet.

openSM2

Die Produktlinie openSM2 ist eine einheitliche Lösung für das unternehmensweite Performance Management von Server- und Speichersystemen. openSM2 bietet eine Messdatenerfassung, Online-Überwachung und Offline-Auswertung.

openUTM-Anwendung
openUTM application

siehe *UTM-Anwendung*.

openUTM-Cluster
openUTM cluster

aus der Sicht von UPIC-Clients, **nicht** aus Server-Sicht:
Zusammenfassung mehrerer Knoten-Anwendungen einer UTM-Cluster-Anwendung zu einer logischen Anwendung, die über einen gemeinsamen Symbolic Destination Name adressiert wird.

openUTM-D

openUTM-D (openUTM-Distributed) ist eine openUTM-Komponente, die *verteilte Verarbeitung* ermöglicht. openUTM-D ist integraler Bestandteil von openUTM.

OSI-LPAP-Bündel
OSI-LPAP bundle

LPAP-Bündel für *OSI TP*-Partner-Anwendungen.

OSI-LPAP-Partner
OSI-LPAP partner

OSI-LPAP-Partner sind die bei openUTM generierten Adressen der *OSI TP-Partner*. Für die *verteilte Verarbeitung* über das Protokoll *OSI TP* muss in der lokalen Anwendung für jede Partner-Anwendung ein OSI-LPAP-Partner konfiguriert werden. Der OSI-LPAP-Partner spiegelt in der lokalen Anwendung die Partner-Anwendung wider. Bei der Kommunikation wird die Partner-Anwendung nicht über ihren Anwendungsnamen oder ihre Adresse, sondern über den Namen des zugeordneten OSI-LPAP-Partners angesprochen.

OSI-Referenzmodell
OSI reference model

Das OSI-Referenzmodell stellt einen Rahmen für die Standardisierung der Kommunikation von offenen Systemen dar. ISO, die Internationale Organisation für Standardisierung, hat dieses Modell im internationalen Standard ISO IS7498 beschrieben. Das OSI-Referenzmodell unterteilt die für die Kommunikation von Systemen notwendigen Funktionen in sieben logische Schichten. Diese Schichten haben jeweils klar definierte Schnittstellen zu den benachbarten Schichten.

OSI TP

Von der ISO definiertes Kommunikationsprotokoll für die verteilte Transaktionsverarbeitung. OSI TP steht für Open System Interconnection Transaction Processing.

OSI TP-Partner
OSI TP partner

Partner der UTM-Anwendung, der mit der UTM-Anwendung über das OSI TP-Protokoll kommuniziert.
Beispiele für solche Partner sind:

- eine UTM-Anwendung, die über OSI TP kommuniziert
- eine Anwendung im IBM-Umfeld (z.B. CICS), die über openUTM-LU62 angeschlossen ist
- eine Anwendung des Trägersystems OpenCPIC des openUTM-Client
- Anwendungen anderer TP-Monitore, die OSI TP unterstützen

Outbound-Conversation (CPI-C)
outbound conversation

siehe *Outgoing-Conversation*.

Outgoing-Conversation (CPI-C)
outgoing conversation

Eine Conversation, bei der das lokale CPI-C-Programm der *Initiator* ist, heißt Outgoing-Conversation. In der X/Open-Specification wird für Outgoing-Conversation auch das Synonym Outbound-Conversation verwendet.

Pagepool
page pool

Teil der *KDCFILE*, in dem Anwenderdaten gespeichert werden.
In einer *stand-alone-Anwendung* sind dies z.B. *Dialog-Nachrichten*, Nachrichten an *Message Queues*, *Sekundärspeicherbereiche*.
In einer UTM-Cluster-Anwendung sind dies z.B. Nachrichten an *Message Queues, TLS*.

Parameterbereich
parameter area
Datenstruktur, in der ein *Teilprogramm* bei einem UTM-Aufruf die für diesen Aufruf notwendigen Operanden an openUTM übergibt.

Postselection (BS2000/OSD)
postselection
Auswahl der protokollierten UTM-Ereignisse aus der SAT-Protokolldatei, die ausgewertet werden sollen. Die Auswahl erfolgt mit Hilfe des Tools SATUT.

Prädialog (BS2000/OSD)
predialog
Aufforderung des Terminal-Benutzers an das Datenkommunikationssystem zum Aufbau einer *logischen Verbindung* zur *Anwendung*. Der Prädialog entfällt, wenn die logische Verbindung auf Initiative der Anwendung aufgebaut wird.

Prepare to commit (PTC)
prepare to commit
Bestimmter Zustand einer verteilten Transaktion:
Das Transaktionsende der verteilten Transaktion wurde eingeleitet, es wird jedoch noch auf die Bestätigung des Transaktionsendes durch den Partner gewartet.

Preselection (BS2000/OSD)
preselection
Festlegung der für die *SAT-Beweissicherung* zu protokollierenden UTM-Ereignisse. Die Preselection erfolgt durch die UTM-SAT-Administration. Man unterscheidet Ereignis-spezifische, Benutzer-spezifische und Auftrags-(TAC-)spezifische Preselection.

Presentation-Selektor
presentation selector
Der Presentation-Selektor identifiziert im lokalen System einen *Dienstzugriffspunkt* zur Darstellungsschicht des *OSI-Referenzmodells*.

Primärspeicherbereich
primary storage area
Bereich im Arbeitsspeicher, auf den das *KDCS-Teilprogramm* direkt zugreifen kann, z.B. *Standard Primärer Arbeitsbereich*, *Kommunikationsbereich*.

Printerprozess (Unix-Systeme)
printer process
siehe *Druckerprozess*.

Programmschnittstelle zur Administration
program interface for administration

UTM-Programmschnittstelle, mit deren Hilfe der Anwender eigene *Administrationsprogramme* erstellen kann. Die Programmschnittstelle zur Administration bietet u.a. Funktionen zur *dynamischen Konfiguration*, zur Modifikation von Eigenschaften und Anwendungsparametern und zur Abfrage von Informationen zur *Konfiguration* und zur aktuellen Auslastung der Anwendung.

Prozess
process

In den openUTM-Handbüchern wird der Begriff „Prozess“ als Oberbegriff für Prozess (Unix-/Windows-Systeme) und Task (BS2000/OSD) verwendet.

Queue
queue

siehe *Message Queue*

Quick Start Kit

Beispielanwendung, die mit openUTM (Windows-Systeme) ausgeliefert wird.

Quittungs-Auftrag
confirmation job

Bestandteil eines *Auftrags-Komplexes*, worin der Quittungs-Auftrag dem *Basis-Auftrag* zugeordnet ist. Es gibt positive und negative Quittungsaufträge. Bei positivem Ergebnis des *Basis-Auftrags* wird der positive Quittungs-Auftrag wirksam, sonst der negative.

Redelivery
redelivery

Erneutes Zustellen einer *Asynchron-Nachricht*, nachdem diese nicht ordnungsgemäß verarbeitet werden konnte, z.B. weil die *Transaktion* zurückgesetzt oder der *Asynchron-Vorgang* abnormal beendet wurde. Die Nachricht wird wieder in die Message Queue eingereiht und lässt sich damit erneut lesen und/oder verarbeiten.

Reentrant-fähiges Programm
reentrant program

Programm, dessen Code durch die Ausführung nicht verändert wird.
Im BS2000/OSD ist dies Voraussetzung dafür, *Shared Code* zu nutzen.

Request
request

Anforderung einer *Service-Funktion* durch einen *Client* oder einen anderen Server.

Requestor
requestor

In XATMI steht der Begriff Requestor für eine Anwendung, die einen Service aufruft.

Resource Manager
resource manager

Resource Manager (RMs) verwalten Datenressourcen. Ein Beispiel für RMs sind Datenbank-Systeme. openUTM stellt aber auch selbst Resource Manager zur Verfügung, z.B. für den Zugriff auf *Message Queues*, lokale Speicherbereiche und Logging-Dateien. Anwendungsprogramme greifen auf RMs über RM-spezifische Schnittstellen zu. Für Datenbank-Systeme ist dies meist SQL, für die openUTM-RMs die Schnittstelle KDCS.

RFC1006

Von IETF (Internet Engineering Task Force) definiertes Protokoll der TCP/IP-Familie zur Realisierung der ISO-Transportdienste (Transportklasse 0) auf TCP/IP-Basis.

RSA

Abkürzung für die Erfinder des RSA-Verschlüsselungsverfahrens Rivest, Shamir und Adleman. Bei diesem Verfahren wird ein Schlüsselpaar verwendet, das aus einem öffentlichen und einem privaten Schlüssel besteht. Eine Nachricht wird mit dem öffentlichen Schlüssel verschlüsselt und kann nur mit dem privaten Schlüssel entschlüsselt werden. Das RSA-Schlüsselpaar wird von der UTM-Anwendung erzeugt.

SAT-Beweissicherung (BS2000/OSD)
SAT audit

Beweissicherung durch die Komponente SAT (Security Audit Trail) des BS2000-Softwareproduktes SECOS.

Sekundärspeicherbereich
secondary storage area

Transaktionsgesicherter Speicherbereich, auf den das KDCS-*Teilprogramm* mit speziellen Aufrufen zugreifen kann. Lokale Sekundärspeicherbereiche (LSSB) sind einem *Vorgang* zugeordnet, auf globale Sekundärspeicherbereiche (GSSB) kann von allen Vorgängen einer *UTM-Anwendung* zugegriffen werden. Weitere Sekundärspeicherbereiche sind der *Terminal-spezifische Langzeitspeicher (TLS)* und der *User-spezifische Langzeitspeicher (ULS)*.

Selektor
selector

Ein Selektor identifiziert im lokalen System einen *Zugriffspunkt* auf die Dienste einer Schicht des *OSI-Referenzmodells*. Jeder Selektor ist Bestandteil der Adresse des Zugriffspunktes.

Semaphor (Unix-/Windows-Systeme)
semaphore

Betriebsmittel auf Unix- und Windows-Systemen, das zur Steuerung und Synchronisation von Prozessen dient.

Server
server

Ein Server ist eine *Anwendung*, die *Services* zur Verfügung stellt. Oft bezeichnet man auch den Rechner, auf dem Server-Anwendungen laufen, als Server.

Server-Seite einer Conversation (CPI-C)
server side of a conversation

Begriff ersetzt durch *Akzeptor*.

Server-Server-Kommunikation
server-server communication

siehe *verteilte Verarbeitung*.

Service Access Point

siehe *Dienstzugriffspunkt.*

Service
service

Services bearbeiten die *Aufträge*, die an eine Server-Anwendung geschickt werden. Ein Service in einer UTM-Anwendung wird auch *Vorgang* genannt und setzt sich aus einer oder mehreren *Transaktionen* zusammen. Ein Service wird über den *Vorgangs-TAC* aufgerufen. Services können von *Clients* oder anderen Services angefordert werden.

Service-gesteuerte Queue
service controlled queue

Message Queue, bei der der Abruf und die Weiterverarbeitung der Nachrichten durch *Services* gesteuert werden. Ein Service muss zum Lesen der Nachricht explizit einen KDCS-Aufruf (DGET) absetzen.
Service-gesteuerte Queues gibt es bei openUTM in den Varianten *USER-Queue*, *TAC-Queue* und *Temporäre Queue*.

Service Routine
service routine
siehe *Teilprogramm*.

Session
session
Kommunikationsbeziehung zweier adressierbarer Einheiten im Netz über das SNA-Protokoll *LU6.1*.

Session-Selektor
session selector
Der Session-Selektor identifiziert im lokalen System einen *Zugriffspunkt* zu den Diensten der Kommunikationssteuerschicht (Session-Layer) des *OSI-Referenzmodells*.

Shared Code (BS2000/OSD)
shared code
Code, der von mehreren Prozessen gemeinsam benutzt werden kann.

Shared Memory
shared memory
Virtueller Speicherbereich, auf den mehrere Prozesse gleichzeitig zugreifen können.

Shared Objects (Unix-/Windows-Systeme)
shared objects
Teile des *Anwendungsprogramms* können als Shared Objects erzeugt werden. Diese werden dynamisch zur Anwendung dazugebunden und können im laufenden Betrieb ausgetauscht werden. Shared Objects werden mit der KDCDEF-Anweisung SHARED-OBJECT definiert.

Sicherungspunkt
synchronization point, consistency point
Ende einer *Transaktion*. Zu diesem Zeitpunkt werden alle in der Transaktion vorgenommenen Änderungen der *Anwendungsinformation* gegen Systemausfall gesichert und für andere sichtbar gemacht. Während der Transaktion gesetzte Sperren werden wieder aufgehoben.

single system image
Unter single system image versteht man die Eigenschaft eines *Clusters*, nach außen hin als eine einziges, in sich geschlossenes System zu erscheinen. Die heterogene Natur des Clusters und die interne Verteilung der Ressourcen im Cluster ist für die Benutzer des Clusters und die Anwendungen, die mit dem Cluster kommunizieren, nicht sichtbar.

SOA

SOA (Service-oriented architecture).
Eine SOA ist ein Konzept für eine Systemarchitektur, in dem Funktionen in Form von wieder verwendbaren, technisch voneinander unabhängigen und fachlich lose gekoppelten *Services* implementiert werden. Services können unabhängig von zugrunde liegenden Implementierungen über Schnittstellen aufgerufen werden, deren Spezifikationen öffentlich und damit vertrauenswürdig sein können. Service-Interaktion findet über eine dafür vorgesehene Kommunikationsinfrastruktur statt.

SOAP

SOAP (Simple Object Access Protocol) ist ein Protokoll, mit dessen Hilfe Daten zwischen Systemen ausgetauscht und Remote Procedure Calls durchgeführt werden können. SOAP stützt sich auf die Dienste anderer Standards, XML zur Repräsentation der Daten und Internet-Protokolle der Transport- und Anwendungsschicht zur Übertragung der Nachrichten.

Socket-Verbindung
socket connection

Transportsystem-Verbindung, die die Socket-Schnittstelle verwendet. Die Socket-Schnittstelle ist eine Standard-Programmschnittstelle für die Kommunikation über TCP/IP.

stand-alone Anwendung
stand-alone application

siehe *stand-alone UTM-Anwendung*.

stand-alone UTM-Anwendung
stand-alone UTM application

Herkömmliche *UTM-Anwendung*, die nicht Bestandteil einer *UTM-Cluster-Anwendung* ist.

Standard Primärer Arbeitsbereich/SPAB (KDCS)
standard primary working area

Bereich im Arbeitsspeicher, der jedem KDCS-*Teilprogramm* zur Verfügung steht. Sein Inhalt ist zu Beginn des Teilprogrammlaufs undefiniert oder mit einem Füllzeichen vorbelegt.

Startformat
start format

Format, das openUTM am Terminal ausgibt, wenn sich ein Benutzer erfolgreich bei der *UTM-Anwendung* angemeldet hat (ausgenommen nach *Vorgangs-Wiederanlauf* und beim Anmelden über *Anmelde-Vorgang*).

statische Konfiguration
static configuration
Festlegen der *Konfiguration* bei der Generierung mit Hilfe des UTM-Tools *KDCDEF*.

SYSLOG-Datei
SYSLOG file
siehe *System-Protokolldatei*.

System-Protokolldatei
system log file
Datei oder Dateigeneration, in die openUTM während des Laufs einer *UTM-Anwendung* alle UTM-Meldungen protokolliert, für die das *Meldungsziel* SYSLOG definiert ist.

TAC
TAC
siehe *Transaktionscode*.

TAC-Queue
TAC queue
Message Queue, die explizit per KDCDEF-Anweisung generiert wird. Eine TAC-Queue ist eine *Service-gesteuerte Queue* und kann unter dem generierten Namen von jedem Service aus angesprochen werden.

Teilprogramm
program unit
UTM-*Services* werden durch ein oder mehrere Teilprogramme realisiert. Die Teilprogramme sind Bestandteile des *Anwendungsprogramms*. Abhängig vom verwendeten API müssen sie KDCS-, XATMI- oder CPIC-Aufrufe enthalten. Sie sind über *Transaktionscodes* ansprechbar. Einem Teilprogramm können mehrere Transaktionscodes zugeordnet werden.

Temporäre Queue
temporary queue
Message Queue, die dynamisch per Programm erzeugt wird und auch wieder per Programm gelöscht werden kann, vgl. *Service-gesteuerte Queue*.

Terminal-spezifischer Langzeitspeicher/TLS (KDCS)
terminal-specific long-term storage
Sekundärspeicher, der einem *LTERM-, LPAP-* oder *OSI-LPAP-Partner* zugeordnet ist und über das Anwendungsende hinaus erhalten bleibt.

Timerprozess (Unix-/Windows-Systeme)
timer process

Prozess, der Aufträge zur Zeitüberwachung von *Workprozessen* entgegennimmt, sie in ein Auftragsbuch einordnet und nach einer im Auftragsbuch festgelegten Zeit den Workprozessen zur Bearbeitung wieder zustellt.

TNS (Unix-/Windows-Systeme)

Abkürzung für den Transport Name Service, der einem Anwendungsnamen einen Transport-Selektor und das Transportsystem zuordnet, über das die Anwendung erreichbar ist.

Tomcat

siehe *Apache Tomcat*

Transaktion
transaction

Verarbeitungsabschnitt innerhalb eines *Services*, für den die Einhaltung der *ACID-Eigenschaften* garantiert wird. Von den in einer Transaktion beabsichtigten Änderungen der *Anwendungsinformation* werden entweder alle konsistent durchgeführt oder es wird keine durchgeführt (Alles-oder-Nichts Regel). Das Transaktionsende bildet einen *Sicherungspunkt*.

Transaktionscode/TAC
transaction code

Name, über den ein *Teilprogramm* aufgerufen werden kann. Der Transaktionscode wird dem Teilprogramm bei der *statischen* oder *dynamischen Konfiguration* zugeordnet. Einem Teilprogramm können auch mehrere Transaktionscodes zugeordnet werden.

Transaktionsrate
transaction rate

Anzahl der erfolgreich beendeten *Transaktionen* pro Zeiteinheit.

Transfer-Syntax
transfer syntax

Bei *OSI TP* werden die Daten zur Übertragung zwischen zwei Rechnersystemen von der lokalen Darstellung in die Transfer-Syntax umgewandelt. Die Transfer-Syntax beschreibt die Daten in einem neutralen Format, das von allen beteiligten Partnern verstanden wird. Jeder Transfer-Syntax muss ein *Object Identifier* zugeordnet sein.

Transport-Selektor
transport selector

Der Transport-Selektor identifiziert im lokalen System einen *Dienstzugriffspunkt* zur Transportschicht des *OSI-Referenzmodells*.

Transportsystem-Anwendung
transport system application

Anwendung, die direkt auf der Transportsystem-Schnittstelle wie z.B. CMX oder Socket aufsetzt. Für den Anschluss von Transportsystem-Anwendungen muss bei der *Konfiguration* als Partnertyp APPLI oder SOCKET angegeben werden. Eine Transportsystem-Anwendung kann nicht in eine *Verteilte Transaktion* eingebunden werden.

TS-Anwendung
TS application

siehe *Transportsystem-Anwendung.*

Typisierter Puffer (XATMI)
typed buffer

Puffer für den Austausch von typisierten und strukturierten Daten zwischen Kommunikationspartnern. Durch diese typisierten Puffer ist die Struktur der ausgetauschten Daten den Partnern implizit bekannt.

UPIC

Trägersystem für UTM-Clients. UPIC steht für Universal Programming Interface for Communication.

UPIC-Client

Bezeichnung für UTM-Clients mit Trägersystem UPIC.

USER-Queue
USER queue

Message Queue, die openUTM jeder Benutzerkennung zur Verfügung stellt. Eine USER-Queue zählt zu den *Service-gesteuerten Queues* und ist immer der jeweiligen Benutzerkennung zugeordnet. Der Zugriff von fremden UTM-Benutzern auf die eigene USER-Queue kann eingeschränkt werden.

User-spezifischer Langzeitspeicher/ULS
user-specific long-term storage

Sekundärspeicher, der einer *Benutzerkennung,* einer *Session* oder einer *Association* zugeordnet ist und über das Anwendungsende hinaus erhalten bleibt.

USLOG-Datei
USLOG file

siehe *Benutzer-Protokolldatei.*

UTM-Anwendung
UTM application

Eine UTM-Anwendung stellt *Services* zur Verfügung, die Aufträge von *Clients* oder anderen Anwendungen bearbeiten. openUTM übernimmt dabei u.a. die Transaktionssicherung und das Management der Kommunikations- und Systemressourcen. Technisch gesehen ist eine UTM-Anwendung eine Prozessgruppe, die zur Laufzeit eine logische Server-Einheit bildet.

UTM-Cluster-Anwendung
UTM cluster application

UTM-Anwendung, die für den Einsatz auf einem *Cluster* generiert ist und die man logisch als **eine** Anwendung betrachten kann.
Physikalisch gesehen besteht eine UTM-Cluster-Anwendung aus mehreren, identisch generierten UTM-Anwendungen, die auf den einzelnen *Knoten* laufen.

UTM-Cluster-Dateien
UTM cluster files

Oberbegriff für alle Dateien, die für den Ablauf einer UTM-Cluster-Anwendung benötigt werden. Dazu gehören folgende Dateien:

- *Cluster-Konfigurationsdatei*
- *Cluster-User-Datei*
- Dateien des *Cluster-Pagepool*
- *Cluster-GSSB-Datei*
- *Cluster-ULS-Datei*
- Dateien des *Cluster-Administrations-Journals**
- *Cluster-Lock-Datei**
- Lock-Datei zur Start-Serialisierung* (nur bei Unix- und Windows-Systemen)

Die mit * gekennzeichneten Dateien werden beim Start der ersten Knoten-Anwendung angelegt, alle anderen Dateien werden bei der Generierung mit KDCDEF erzeugt.

UTM-D

siehe *openUTM-D.*

UTM-Datenstation
UTM terminal

Begriff ersetzt durch *LTERM-Partner*.

UTM-F

UTM-Anwendungen können als UTM-F-Anwendungen (UTM-Fast) generiert werden. Bei UTM-F wird zugunsten der Performance auf Platteneingaben/-ausgaben verzichtet, mit denen bei *UTM-S* die Sicherung von Benutzer- und Transaktionsdaten durchgeführt wird. Gesichert werden lediglich Änderungen der Verwaltungsdaten.
In UTM-Cluster-Anwendungen, die als UTM-F-Anwendung generiert sind (APPLIMODE=FAST), werden Cluster-weit gültige Anwenderdaten auch gesichert. Dabei werden GSSB- und ULS-Daten genauso behandelt wie in UTM-Cluster-Anwendungen, die mit UTM-S generiert sind. Vorgangs-Daten von Benutzern mit RESTART=YES werden jedoch nur beim Abmelden des Benutzers anstatt bei jedem Transaktionsende geschrieben.

UTM-gesteuerte Queues
UTM controlled queue

Message Queues, bei denen der Abruf und die Weiterverarbeitung der Nachrichten vollständig durch openUTM gesteuert werden. Siehe auch *Asynchron-Auftrag, Hintergrund-Auftrag* und *Asynchron-Nachricht*.

UTM-S

Bei UTM-S-Anwendungen sichert openUTM neben den Verwaltungsdaten auch alle Benutzerdaten über ein Anwendungsende und einen Systemausfall hinaus. Außerdem garantiert UTM-S bei allen Störungen die Sicherheit und Konsistenz der Anwendungsdaten. Im Standardfall werden UTM-Anwendungen als UTM-S-Anwendungen (UTM-Secure) generiert.

UTM-SAT-Administration (BS2000/OSD)
UTM SAT administration

Durch die UTM-SAT-Administration wird gesteuert, welche sicherheitsrelevanten UTM-Ereignisse, die im Betrieb der *UTM-Anwendung* auftreten, von *SAT* protokolliert werden sollen. Für die UTM-SAT-Administration wird eine besondere Berechtigung benötigt.

UTM-Seite
UTM page

Ist eine Speichereinheit, die entweder 2K oder 4K umfasst. In *stand-alone UTM-Anwendungen* kann die Größe einer UTM-Seite bei der Generierung der UTM-Anwendung auf 2K oder 4K gesetzt werden. In einer *UTM-Cluster-Anwendung* ist die Größe einer UTM-Seite immer 4K. *Pagepool* und Wiederanlauf-Bereich der KDCFILE sowie *UTM-Cluster-Dateien* werden in Einheiten der Größe einer UTM-Seite unterteilt.

utmpfad (Unix-/Windows-Systeme)
utmpath

Das Dateiverzeichnis unter dem die Komponenten von openUTM installiert sind, wird in diesem Handbuch als *utmpfad* bezeichnet.
Um einen korrekten Ablauf von openUTM zu garantieren, muss die Umgebungsvariable UTMPATH auf den Wert von *utmpfad* gesetzt werden. Auf Unix-Systemen müssen Sie UTMPATH vor dem Starten einer UTM-Anwendung setzen, auf Windows-Systemen wird UTMPATH bei der Installation gesetzt.

Verarbeitungsschritt
processing step

Ein Verarbeitungsschritt beginnt mit dem Empfangen einer *Dialog-Nachricht*, die von einem *Client* oder einer anderen Server-Anwendung an die *UTM-Anwendung* gesendet wird. Der Verarbeitungsschritt endet entweder mit dem Senden einer Antwort und beendet damit auch den *Dialog-Schritt* oder er endet mit dem Senden einer Dialog-Nachricht an einen Dritten.

Verbindungs-Benutzerkennung
connection user ID

Benutzerkennung, unter der eine *TS-Anwendung* oder ein *UPIC-Client* direkt nach dem Verbindungsaufbau bei der *UTM-Anwendung* angemeldet wird.
Abhängig von der Generierung des Clients (= LTERM-Partner) gilt:

- Die Verbindungs-Benutzerkennung ist gleich dem USER der LTERM-Anweisung (explizite Verbindungs-Benutzerkennung). Eine explizite Verbindungs-Benutzerkennung muss mit einer USER-Anweisung generiert sein und kann nicht als „echte" *Benutzerkennung* verwendet werden.
- Die Verbindungs-Benutzerkennung ist gleich dem LTERM-Partner (implizite Verbindungs-Benutzerkennung), wenn bei der LTERM-Anweisung kein USER angegeben wurde oder wenn ein LTERM-Pool generiert wurde.

In einer *UTM-Cluster-Anwendung* ist der Vorgang einer Verbindungs-Benutzerkennung (RESTART=YES bei LTERM oder USER) an die Verbindung gebunden und damit Knoten-lokal.
Eine Verbindungs-Benutzerkennung, die mit RESTART=YES generiert ist, kann in jeder *Knoten-Anwendung* einen eigenen Vorgang haben.

Verbindungsbündel
connection bundle

siehe *LTERM-Bündel*.

Verschlüsselungsstufe
encryption level

Die Verschlüsselungsstufe legt fest, ob und inwieweit ein Client Nachrichten und Passwort verschlüsseln muss.

Verteilte Transaktion
distributed transaction

Transaktion, die sich über mehr als eine Anwendung erstreckt und in mehreren (Teil)-Transaktionen in verteilten Systemen ausgeführt wird.

Verteilte Transaktionsverarbeitung
Distributed Transaction Processing

Verteilte Verarbeitung mit *verteilten Transaktionen*.

Verteilte Verarbeitung
distributed processing

Bearbeitung von *Dialog-Aufträgen* durch mehrere Anwendungen oder Übermittlung von *Hintergrundaufträgen* an eine andere Anwendung. Für die verteilte Verarbeitung werden die höheren Kommunikationsprotokolle *LU6.1* und *OSI TP* verwendet. Über openUTM-LU62 ist verteilte Verarbeitung auch mit LU6.2 Partnern möglich. Man unterscheidet verteilte Verarbeitung mit *verteilten Transaktionen* (Anwendungs-übergreifende Transaktionssicherung) und verteilte Verarbeitung ohne verteilte Transaktionen (nur lokale Transaktionssicherung). Die verteilte Verarbeitung wird auch Server-Server-Kommunikation genannt.

Vorgang (KDCS)
service

Ein Vorgang dient zur Bearbeitung eines *Auftrags* in einer *UTM-Anwendung*. Er setzt sich aus einer oder mehreren *Transaktionen* zusammen. Die erste Transaktion wird über den *Vorgangs-TAC* aufgerufen. Es gibt *Dialog-Vorgänge* und *Asynchron-Vorgänge*. openUTM stellt den Teilprogrammen eines Vorgangs gemeinsame Datenbereiche zur Verfügung. Anstelle des Begriffs Vorgang wird häufig auch der allgemeinere Begriff *Service* gebraucht.

Vorgangs-Kellerung (KDCS)
service stacking

Ein Terminal-Benutzer kann einen laufenden *Dialog-Vorgang* unterbrechen und einen neuen Dialog-Vorgang einschieben. Nach Beendigung des eingeschobenen *Vorgangs* wird der unterbrochene Vorgang fortgesetzt.

Vorgangs-Kettung (KDCS)
service chaining

Bei Vorgangs-Kettung wird nach Beendigung eines *Dialog-Vorgangs* ohne Angabe einer *Dialog-Nachricht* ein Folgevorgang gestartet.

Vorgangs-TAC (KDCS)
service TAC

Transaktionscode, mit dem ein *Vorgang* gestartet wird.

Vorgangs-Wiederanlauf (KDCS)
service restart

Wird ein Vorgang unterbrochen, z.B. infolge Abmeldens des Terminal-Benutzers oder Beendigung der *UTM-Anwendung*, führt openUTM einen Vorgangs-Wiederanlauf durch. Ein *Asynchron-Vorgang* wird neu gestartet oder beim zuletzt erreichten *Sicherungspunkt* fortgesetzt, ein *Dialog-Vorgang* wird beim zuletzt erreichten Sicherungspunkt fortgesetzt. Für den Terminal-Benutzer wird der Vorgangs-Wiederanlauf eines Dialog-Vorgangs als *Bildschirm-Wiederanlauf* sichtbar, sofern am letzten Sicherungspunkt eine Dialog-Nachricht an den Terminal-Benutzer gesendet wurde.

Warmstart
warm start

Start einer *UTM-S*-Anwendung nach einer vorhergehenden abnormalen Beendigung. Dabei wird die *Anwendungsinformation* auf den zuletzt erreichten konsistenten Zustand gesetzt. Unterbrochene *Dialog-Vorgänge* werden dabei auf den zuletzt erreichten *Sicherungspunkt* zurückgesetzt, so dass die Verarbeitung an dieser Stelle wieder konsistent aufgenommen werden kann (*Vorgangs-Wiederanlauf*). Unterbrochene *Asynchron-Vorgänge* werden zurückgesetzt und neu gestartet oder beim zuletzt erreichten *Sicherungspunkt* fortgesetzt.
Bei UTM-F-Anwendungen werden beim Start nach einer vorhergehenden abnormalen Beendigung lediglich die dynamisch geänderten Konfigurationsdaten auf den zuletzt erreichten konsistenten Zustand gesetzt.
In UTM-Cluster-Anwendungen werden die globalen Sperren auf GSSB und ULS, die bei der abnormalen Beendigung von dieser Knoten-Anwendung gehalten wurden, aufgehoben. Außerdem werden Benutzer, die zum Zeitpunkt der abnormalen Benendigung an dieser Knoten-Anwendung angemeldet waren, abgemeldet.

Web Service
web service

Anwendung, die auf einem Web-Server läuft und über eine standardisierte und programmatische Schnittstelle (öffentlich) verfügbar ist. Die Web Services-Technologie ermöglicht es, UTM-Teilprogramme für moderne Web-Client-Anwendungen verfügbar zu machen, unabhängig davon, in welcher Programmiersprache sie entwickelt wurden.

Wiederanlauf
restart

siehe *Bildschirm-Wiederanlauf*,
siehe *Vorgangs-Wiederanlauf.*

Workprozess (Unix-/Windows-Systeme)
work process

Prozess, in dem die *Services* der *UTM-Anwendung* ablaufen.

WS4UTM

WS4UTM (**W**eb**S**ervices for open**UTM**) ermöglicht es Ihnen, auf komfortable Weise einen Service einer UTM-Anwendung als Web Service zur Verfügung zu stellen.

XATMI

XATMI (X/Open Application Transaction Manager Interface) ist eine von X/Open standardisierte Programmschnittstelle für die Programm-Programm-Kommunikation in offenen Netzen.
Das in openUTM implementierte XATMI genügt der XATMI CAE Specification von X/Open. Die Schnittstelle steht in COBOL und C zur Verfügung. XATMI in openUTM kann über die Protokolle *OSI TP*, *LU6.1* und *UPIC* kommunizieren.

XHCS (BS2000/OSD)

XHCS (Extended Host Code Support) ist ein BS2000/OSD-Softwareprodukt für die Unterstützung internationaler Zeichensätze.

XML

XML (eXtensible Markup Language) ist eine vom W3C (WWW-Konsortium) genormte Metasprache, in der Austauschformate für Daten und zugehörige Informationen definiert werden können.

Zeitgesteuerter Auftrag
time-driven job

Auftrag, der von openUTM bis zu einem definierten Zeitpunkt in einer *Message Queue* zwischengespeichert und dann an den Empfänger weitergeleitet wird. Empfänger kann sein: ein *Asynchron-Vorgang* der selben Anwendung, eine *TAC-Queue*, eine Partner-Anwendung, ein Terminal oder ein Drucker.
Zeitgesteuerte Aufträge können nur von KDCS-*Teilprogrammen* erteilt werden.

Zugangskontrolle
system access control

Prüfung durch openUTM, ob eine bestimmte *Benutzerkennung* berechtigt ist, mit der *UTM-Anwendung* zu arbeiten. Die Berechtigungsprüfung entfällt, wenn die UTM-Anwendung ohne Benutzerkennungen generiert wurde.

Zugriffskontrolle
data access control

Prüfung durch openUTM, ob der Kommunikationspartner berechtigt ist, auf ein bestimmtes Objekt der Anwendung zuzugreifen. Die Zugriffsrechte werden als Bestandteil der Konfiguration festgelegt.

Zugriffspunkt
access point

siehe *Dienstzugriffspunkt*.

Abkürzungen

ACSE	Association Control Service Element
AEQ	Application Entity Qualifier
AES	Advanced Encryption Standard
AET	Application Entity Title
APT	Application Process Title
ASCII	American Standard Code for Information Interchange
ASE	Application Service Element
Axis	Apache eXtensible Interaction System
BCAM	Basic Communication Access Method
BER	Basic Encoding Rules
BLS	Binder-Lader-Starter (BS2000/OSD)
CCP	Communication Control Program
CCR	Commitment, Concurrency and Recovery
CCS	Codierter Zeichensatz (Coded Character Set)
CCSN	Name des codierten Zeichensatzes (Coded Character Set Name)
CICS	Customer Information Control System (IBM)
CID	Control Identification
CMX	Communication Manager in Unix-Systemen
COM	Component Object Model
CPI-C	Common Programming Interface for Communication
CRM	Communication Resource Manager
CRTE	Common Runtime Environment (BS2000/OSD)
DB	Datenbank
DC	Data Communication
DCAM	Data Communication Access Method

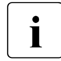
DCOM	Distributed Component Object Mode
DES	Data Encryption Standard
DLM	Distributed Lock Manager (BS2000/OSD)
DMS	Data Management System
DNS	Domain Name Service
DSS	Datensichtstation (=Terminal)
DTD	Document Type Definition
DTP	Distributed Transaction Processing
DVS	Datenverwaltungssystem
EBCDIC	Binärcode für die stellenweise Verschlüsselung von Dezimalziffern (Extended Binary-Coded Decimal Interchange Code)
EJB	Enterprise JavaBeans^{TM}
FGG	File Generation Group
FHS	Format Handling System
FT	File Transfer
GSSB	Globaler Sekundärer Speicherbereich
HIPLEX®	Highly Integrated System Complex (BS2000/OSD)
HLL	High-Level Language
HTML	Hypertext Markup Language
HTTP	Hypertext Transfer Protocol
IFG	Interaktiver Format-Generator
ILCS	Inter Language Communication Services (BS2000/OSD)
IMS	Information Management System (IBM)
IPC	Inter-Process-Communication
IRV	Internationale Referenzversion
ISO	International Organization for Standardization
J2EE	Java 2 Enterprise Edition Technologie
JCA	Java Connector Architecture
JDK	Java Development Kit
JEE5	Java Enterprise Edition 5.0
KAA	KDCS Application Area
KB	Kommunikationsbereich
KBPROG	KB-Programmbereich

KDCS	Kompatible Datenkommunikationsschnittstelle
KTA	KDCS Task Area
LAN	Local Area Network
LCF	Local Configuration File
LLM	Link and Load Module (BS2000/OSD)
LSSB	Lokaler Sekundärer Speicherbereich
LU	Logical Unit
MIGRAT	Migrationsprogramm
MQ	Message Queuing
MSCF	Multiple System Control Facility (BS2000/OSD)
NB	Nachrichtenbereich
NEA	Netzwerkarchitektur bei TRANSDATA-Systemen
NFS	Network File System/Service
NLS	Unterstützung der Landessprache (Native Language Support)
OCX	OLE Control Extension
OLTP	Online Transaction Processing
OML	Object Modul Library
OSI	Open System Interconnection
OSI TP	Open System Interconnection Transaction Processing
OSS	OSI Session Service
PCMX	Portable Communication Manager
PDN	Programmsystem für Datenfernverarbeitung und Netzsteuerung
PID	Prozess-Identifikation
PIN	Persönliche Identifikationsnummer
PLU	Primary Logical Unit
PTC	Prepare to commit
RAV	Rechenzentrums-Abrechnungs-Verfahren
RDF	Resource Definition File
RM	Resource Manager
RSA	Encryption-Algorithmus nach Rivest, Shamir, Adleman
RSO	Remote SPOOL Output (BS2000/OSD)
RTS	Runtime System (Laufzeitsystem)

SAT	Security Audit Trail (BS2000/OSD)
SECOS	Security Control System
SGML	Standard Generalized Markup Language
SLU	Secondary Logical Unit
SM2	Software Monitor 2 (BS2000/OSD)
SNA	Systems Network Architecture
SOA	Service-oriented Architecture
SOAP	Simple Object Access Protocol
SPAB	Standard Primärer Arbeitsbereich
SQL	Structured Query Language
SSB	Sekundärer Speicherbereich
SSO	Single-Sign-On
TAC	Transaktionscode
TCEP	Transport Connection End Point
TCP/IP	Transport Control Protocol / Internet Protocol
TIAM	Terminal Interactive Access Method
TLS	Terminal-spezifischer Langzeitspeicher
TM	Transaction Manager
TNS	Transport Name Service
TP	Transaction Processing (Transaktions-Betrieb)
TPR	Task privileged (privilegierter Funktionszustand des BS2000/OSD)
TPSU	Transaction Protocol Service User
TSAP	Transport Service Access Point
TSN	Task Sequence Number
TU	Task user (nicht privilegierter Funktionszustand des BS2000/OSD)
TX	Transaction Demarcation (X/Open)
UDDI	Universal Description, Discovery and Integration
UDS	Universelles Datenbanksystem
UDT	Unstructured Data Transfer
ULS	User-spezifischer Langzeitspeicher
UPIC	Universal Programming Interface for Communication
USP	UTM-Socket-Protokoll
UTM	Universeller Transaktionsmonitor

UTM-D	UTM-Funktionen für verteilte Verarbeitung („Distributed“)
UTM-F	Schnelle UTM-Variante („Fast“)
UTM-S	UTM-Sicherheitsvariante
UTM-XML	XML-Schnittstelle von openUTM
VGID	Vorgangs-Identifikation
VTSU	Virtual Terminal Support
VTV	Verteilte Transaktionsverarbeitung
VV	Verteilte Verarbeitung
WAN	Wide Area Network
WS4UTM	WebServices for openUTM
WSDD	Web Service Deployment Descriptor
WSDL	Web Services Description Language
XA	X/Open Access Interface (Schnittstelle von X/Open zum Zugriff auf Resource Manager)
XAP	X/OPEN ACSE/Presentation programming interface
XAP-TP	X/OPEN ACSE/Presentation programming interface Transaction Processing extension
XATMI	X/Open Application Transaction Manager Interface
XCS	Cross Coupled System
XHCS	eXtended Host Code Support
XML	eXtensible Markup Language

Literatur



PDF-Dateien von allen openUTM-Handbüchern sind sowohl auf der Enterprise Edition DVD für die offenen Plattformen als auch auf der openUTM WinAdmin-DVD (für BS2000/OSD) enthalten.

Alle Handbücher sind online unter *http://manuals.ts.fujitsu.com* zu finden.

Dokumentation zu openUTM

openUTM
Konzepte und Funktionen
Benutzerhandbuch

openUTM
Anwendungen programmieren mit KDCS für COBOL, C und C++
Basishandbuch

openUTM
Anwendungen generieren
Benutzerhandbuch

openUTM
Einsatz von openUTM-Anwendungen unter BS2000/OSD
Benutzerhandbuch

openUTM
Einsatz von openUTM-Anwendungen unter Unix- und Windows-Systemen
Benutzerhandbuch

openUTM
Anwendungen administrieren
Benutzerhandbuch

openUTM
Meldungen, Test und Diagnose in BS2000/OSD
Benutzerhandbuch

openUTM
Meldungen, Test und Diagnose in Unix- und Windows-Systemen
Benutzerhandbuch

openUTM (BS2000/OSD, Unix-Systeme, Windows NT)
Anwendungen erstellen mit X/Open-Schnittstellen
Basishandbuch

openUTM
XML für openUTM

openUTM-Client (Unix-Systeme)
für Trägersystem OpenCPIC
Client-Server-Kommunikation mit openUTM
Benutzerhandbuch

openUTM WinAdmin
Grafischer Administrationsarbeitsplatz für openUTM
Beschreibung und Online-Hilfe

openUTM, **openUTM-LU62**
Verteilte Transaktionsverarbeitung
zwischen openUTM und CICS-, IMS- und LU6.2-Anwendungen
Benutzerhandbuch

openUTM (BS2000/OSD)
Anwendungen programmieren mit KDCS für Assembler
Ergänzung zum Basishandbuch

openUTM (BS2000/OSD)
Anwendungen programmieren mit KDCS für Fortran
Ergänzung zum Basishandbuch

openUTM (BS2000/OSD)
Anwendungen programmieren mit KDCS für Pascal-XT
Ergänzung zum Basishandbuch

openUTM (BS2000/OSD)
Anwendungen programmieren mit KDCS für PL/I
Ergänzung zum Basishandbuch

WS4UTM (Unix- und Windows-Systeme)
Web-Services für openUTM

openUTM
Masterindex

Dokumentation zum openSEAS-Produktumfeld

BeanConnect
Benutzerhandbuch

JConnect
Verbindung von Java-Clients zu openUTM
Benutzerdokumentation und Java-Docs

WebTransactions
Konzepte und Funktionen

WebTransactions
Template-Sprache

WebTransactions
Anschluss an openUTM-Anwendungen über UPIC

WebTransactions
Anschluss an MVS-Anwendungen

WebTransactions
Anschluss an OSD-Anwendungen

Dokumentation zum BS2000/OSD-Umfeld

i Die meisten dieser Handbücher sind zusätzlich in gedruckter Form gegen gesondertes Entgelt unter *http://manualshop.ts.fujitsu.com* zu bestellen.

AID (BS2000/OSD)
Advanced Interactive Debugger
Basishandbuch
Benutzerhandbuch

BCAM (BS2000/OSD)
BCAM Band 1/2
Benutzerhandbuch

BINDER (BS2000/OSD)
Benutzerhandbuch

BS2000/OSD
Makroaufrufe an den Ablaufteil
Benutzerhandbuch

BS2000/OSD-BC
BLSSERV
Bindelader-Starter
Benutzerhandbuch

DCAM (BS2000/OSD)
COBOL-Aufrufe
Benutzerhandbuch

DCAM (BS2000/OSD)
Makroaufrufe
Benutzerhandbuch

DCAM (BS2000/OSD)
Programmschnittstellen
Beschreibung

FHS (BS2000/OSD)
Formatierungssystem für openUTM, TIAM, DCAM
Benutzerhandbuch

IFG für FHS
Benutzerhandbuch

FHS-DOORS (BS2000/OSD, MS-Windows)
Grafische Oberfläche für BS2000/OSD-Anwendungen
Benutzerhandbuch

HIPLEX AF (BS2000/OSD)
Hochverfügbarkeit von Anwendungen in BS2000/OSD
Produkthandbuch

HIPLEX MSCF (BS2000/OSD)
BS2000-Rechner im Verbund
Benutzerhandbuch

IMON (BS2000/OSD)
Installationsmonitor
Benutzerhandbuch

MT9750 (MS Windows)
9750-Emulation unter Windows
Produkthandbuch

OMNIS/OMNIS-MENU (BS2000)
Funktionen und Kommandos
Benutzerhandbuch

OMNIS/OMNIS-MENU (BS2000)
Administration und Programmierung
Benutzerhandbuch

OMNIS-MENU (BS2000/OSD)
Benutzerhandbuch

OSS (BS2000/OSD)
OSI Session Service
User Guide

RSO (BS2000/OSD)
Remote SPOOL Output
Benutzerhandbuch

SECOS (BS2000/OSD)
Security Control System
Benutzerhandbuch

SECOS (BS2000/OSD)
Security Control System
Tabellenheft

SESAM/SQL (BS2000/OSD)
Datenbankbetrieb
Benutzerhandbuch

openSM2
Software Monitor
Band 1: Verwaltung und Bedienung

TIAM (BS2000/OSD)
Benutzerhandbuch

UDS/SQL (BS2000/OSD)
Datenbankbetrieb
Benutzerhandbuch

Unicode im BS2000/OSD
Übersichtshandbuch

VTSU (BS2000/OSD)
Virtual Terminal Support
Benutzerhandbuch

XHCS (BS2000/OSD)
8-bit-Code- und Unicode-Unterstützung im BS2000/OSD
Benutzerhandbuch

Dokumentation zum Umfeld von Unix-Systemen

CMX V6.0 (Solaris)
Betrieb und Administration
Benutzerhandbuch

CMX V6.0 (Unix-Systeme)
Betrieb und Administration
Benutzerhandbuch

CMX V6.0
CMX-Anwendungen programmieren
Programmierhandbuch

OSS (SINIX)
OSI Session Service
User Guide

PRIMECLUSTER*TM*
Konzept (Solaris, Linux)
Benutzerhandbuch

openSM2
Die Dokumentation zu openSM2 wird in Form von ausführlichen Online-Hilfen bereitgestellt, die mit dem Produkt ausgeliefert werden.

Sonstige Literatur

CPI-C (X/Open)
Distributed Transaction Processing
X/Open CAE Specification, Version 2
ISBN 1 85912 135 7

Reference Model Version 2 (X/Open)
Distributed Transaction Processing
X/Open Guide
ISBN 1 85912 019 9

TX (Transaction Demarcation) (X/Open)
Distributed Transaction Processing
X/Open CAE Specification
ISBN 1 85912 094 6

XATMI (X/Open)
Distributed Transaction Processing
X/Open CAE Specification
ISBN 1 85912 130 6

XML
Spezifikation des W3C (www – Konsortium)
Webseite: http://www.w3.org/XML

Stichwörter

#Format 65
*Format 65
+Format 65
-Format 65
.DEFAULT 304, 311

19Z, MGET 211

A
Abfragen
  CHARACTER_CONVERTION 125
  Client-Kontext 115
  erweiterte Information 131
  Offset der Cursor-Position 127
  Verschlüsselungsebene 120
  Zustand der Conversation 123
Ablauf Receive-Timer 158, 170
Abmelden
  CPI-C-Programm 109
  XATMI 260
  XATMI-Client 263
Abnormale Beendigung
  Conversation 105
  CPI-C-Programm 333
Access-List-Konzept 79
Administrations-Journal 375
Adressierung
  CPI-C 54
Adressierungsformate 293
Adressierungsinformationen 294
  für das Netzwerk 295
  von TCP/IP 294
Advanced Functions 98, 218
AES-Schlüssel 84
Aktive RSA Schlüssel 85
Allocate-Aufruf 100
Anbindung über UPIC-Local 184
Ändern
  Senderichtung 162
Anmelden
  bei abgelaufenem Passwort 80
  CPI-C-Programm 111
  mehrfach bei UTM 113
  XATMI 260
Anschluss ans Trägersystem (XATMI) 260
ANSI 282, 321, 356
ANSI-Compiler 356
Anwendung
  Port für lokale Anwendung 238
Anzeigen Senderecht
  Receive 155
  Receive_Mapped_Data 167
applifile 149
ASCII 294
ASCII nach EBCDIC Konvertierung (CPI-C) 68, 104
ASCII-Konvertierung (XATMI) 258
ASCII-Zeichen 71
ASN.1-Typ 257
Asynchrones Request-Response-Modell 254
Attributfelder, Format 65
Aufbau
  einer CPI-C-Anwendung 55
Aufbereiten
  UPIC-Trace 343
Aufruf
  Advanced Functions (CPI-C) 98
  Allocate 100

Aufruf (Forts.)
Convert_Incoming 103
Convert_Outgoing 104
Deallocate 105
Deferred_Deallocate 107
Disable_UTM_UPIC 109
Enable_UTM_UPIC 111
Extract_Conversation_State 123
Extract_Cursor_Offset 127
Extract_Partner_LU_Name 129
Extract_Secondary_Information 131
Initialize_Conversation 147
Prepare_To_Receive 151
Receive 154
Send_Mapped_Data 181
Set_Allocate_Timer 184
Set_Conversation_Encryption_Level 191
Set_Conversation_New_Password 195
Set_Conversation_Security_Password 198
Set_Conversation_Security_Type 201
Set_Conversation_Security_User_ID 203
Set_Deallocate_Type 208
Set_Function_Key 210
Set_Partner_Host_Name 213
Set_Partner_IP_Address 215
Set_Partner_LU_Name 218
Set_Partner_Port 221
Set_Partner_Tsel 223
Set_Partner_Tsel_Format 225
Set_Receive_Timer 228
Set_Receive_Type 231
Set_Sync_Level 234
Set_TP_Name 236
Specify_Local_Port 238
Specify_Local_Tsel 240
Specify_Local_Tsel_Format 242
Starter-Set 97
UPIC-Funktionen 99
xatmigen 276
Ausgabeparameter (CPI-C) 96
Ausrichtung in typisierten Puffern 256
Ausschalten UPIC-Trace 343
Automatische Konvertierung 69
festlegen 147, 298

B
BADTAC 332
BCMAP 296
Bearbeiten
Konvertierungstabelle 69
Beenden
Thread 90
Beenden Conversation
abnormal 105, 208
nach Transaktionsende 107
Beendete UTM-Anwendung 333
Beispiel
Client-Anbindung generieren 349
Multiple Conversations 92
Side Information-Datei 346
TNS-Eintrag 346
tpcall (Windows) 347
uptac (Windows) 346
Beispiele für Windows 345
Benutzerdaten
verschlüsseln 84
Benutzerkennung
Mehrfachanmeldungen 80
ungültig 156, 168, 204
Benutzerkonzept von openUTM 77
Betriebsmittel
CPI-C-Programm (Unix-System) 328
CPI-C-Programm (Windows) 323
lokal 336
Partner 336
UPIC-Local (Unix-System) 329
Betriebsmittelengpass, Fehler bei 100
Betriebssystem 31
Betriebssystem-Plattformen 27
Big Endian 256
Binden
BS2000 330
CPI-C-Programm (Unix-System) 327
CPI-C-Programm (Windows) 321
UPIC-Local (Unix-System) 329
XATMI-Programm 282
Blockierender Receive
Receive 154
Receive_Mapped_Data 166

Blockierender Receive(Forts.)
setzen 231
Timer 228
Überblick 60
BOOL Peek 46
BS2000
Ablaufumgebung 331
BCMAP 296
Binden 330
Code-Konvertierung 68
Code-Konvertierung incoming 103
Code-Konvertierung outgoing 104
Code-Konvertierung per upicfile 299
ftp 336
Jobvariable 320
Logging-Datei 338
partner_LU_name 299
ping 335
telnet 336
Übersetzen 330
UPIC-Trace 339
upicfile 297
XATMI aufrufen 276
buffer
Extract_Shutdown_Time 141
Receive 155
Receive_Mapped_Data 167
Send_Data 178
Send_Mapped_Data 181
BUFFER-Anweisung 274

C
C++ Class CUpic 33
C-Datentypen 256
C-Source 70
CD 306
CHARACTER_CONVERTION 53, 69
setzen 206
Characteristic PROTOCOL
abfragen 118
setzen 189
Characteristics einer Conversation (CPI-C) 50
Charakteristika (XATMI) 269
ClassCUpic 40
Client 25
Initialisieren (XATMI) 261
XATMI 249
Client-Anbindung, Generierungsbeispiele 349
Client-Kontext 83
abfragen 115
Client-Server-Konzept 25
CLIENT_CONTEXT 53
Cluster 32
Cluster-Administrations-Journal 375
Cluster-Anwendung 32
CM_COMMUNICATION_PROTOCOL_34
abfragen 118
setzen im Programm 189
CM_COMMUNICATION_PROTOCOL_40
abfragen 118
setzen im Programm 189
CM_DEALLOCATED_ABEND 333
PEND ER/FR 335
CM_RECEIVE_AND_WAIT 154, 231
CM_RECEIVE_IMMEDIATE 154, 232
CM_SECURITY_PWD_EXPIRED_RETRY 80
CMALLC 100
CMCNVI 103
CMCNVO 104
CMCOBOL 246
CMDEAL 105
CMDFDE 107
CMDISA 109
CMECEL 120
CMECO 127
CMECP 118
CMECS 123
CMENAB 111
CMEPLN 129
CMESI 131
CMESRC 134
CMINIT 147
CMPTR 151
CMRCV 154
CMRCVM 166
CMSAT 184
CMSCC 186
CMSCEL 191

CMSCP 189
CMSCSN 195
CMSCSP 198
CMSCST 201
CMSCSU 203
CMSDT 208
CMSEND 178
CMSFK 210
CMSLP 238
CMSLT 240
CMSLTF 242
CMSNDM 181
CMSPHN 213
CMSPIA 215
CMSPLN 218
CMSPP 221
CMSPT 223
CMSPTF 225
CMSRCT 228
CMSRT 231
CMSSL 234
CMSSRC 244
CMSTPN 236
CMX 30
COBOL-Schnittstelle
  CPI-C 246
  XATMI 259
Code für Datentypen (XATMI) 257, 274
Code-Konvertierung 68
  automatische 298
  für Windows 70
Communication Resource Manager (CRM) 249
COMMUNICATION_PROTOCOL 53
COMP-Anweisung 246
Compileroption
  UTM_ON_WIN32 321, 355
  UTM_UPIC_V11 355
Compilieren
  CPI-C-Programm (Unix-System) 327
  CPI-C-Programm (Windows) 321
CONNECT-MODE 314
Conversation 50, 255
  bedingungslos beenden 208
  beenden 105, 107
  einrichten 100
  einrichten (Standardwerte) 147
  parallele 89
  Zustand 50
  Zustand abfragen 123
Conversation Characteristic 50
  conversation_type 51
  deallocate_type 52
  deallocate_type (Set_Deallocate_Type) 208
  ENCRYPTION-LEVEL 53
  HOSTNAME 53
  IP-ADRESS 53
  partner_LU_name 52, 147
  partner_LU_name_length 52
  partner_LU_name_lth 147
  PORT 53
  receive_type 52, 154
  receive_type (Receive) 154
  receive_type (Receive_Mapped_Data) 166
  receive_type (Set_Receive_Type) 231
  return_control 51
  RSA-KEY 53
  security_new_password 52
  security_new_password_lenght 52
  security_password 52, 198
  security_password_length 52, 198
  security_type 52, 198, 201
  security_user_ID 52, 201, 203
  security_user_ID_length 52, 203
  send_type 51
  Standardwerte 147
  status_received (Send_Data) 178
  status_received (Send_Mapped_Data) 181
  sync_level 51, 234
  T-SEL 53
  T-SEL-FORMAT 53
  TP_name 52, 147
  TP_name (Set_TP_Name) 236
  TP_name_length 52, 236
  Überblick 51
  UPIC-spezifisch 53
  veränderbar 52
  vorgegeben 51

Conversation-ID
ermitteln 148
freigeben 105
conversation_ID (Initialize_Conversation) 148
conversation_type 51
Conversational Modell 255
konfigurieren 273
Convert_Incoming
Aufruf 103
Überblick 69
Convert_Outgoing
Aufruf 104
Überblick 69
copy-cobol85 246
COPY-Element (CPI-C) 246
core 334
CPI-C 13
Einschalung 26
Version 2 354
CPI-C-Anwendung 27
Aufbau 55
CPI-C-Aufrufe 96
C 96
COBOL 246
Reihenfolge 55
CPI-C-Begriffe, Definition 50
CPI-C-Programm 27
abmelden 109
anmelden 111
beendet 333
binden (Unix-System) 327
binden (Windows) 321
Portierbarkeit 235
starten (Unix-System) 328
starten (Windows) 321
Zugriff auf Services 28
CPI-C-Schnittstelle 96
CUPIC Objekt 26
CUpic Security 39
CUpicLocAddr 33, 35
CUpicRemAddr 33, 37
Cursor-Position 68
Offset abfragen 127

D
Darstellungsmittel 22
data_received
Receive 155
Receive_Mapped_Data 167
Daten
empfangen 154
empfangen mit Formatkennzeichen 166
Länge beim Senden 178
senden 178
senden mit Formatkennzeichen 181
Datenbankzugriffe 58
Datenempfang
anzeigen (Receive) 155
anzeigen (Receive_Mapped_Data) 167
Shutdown-Time 142
Datenmengen, große (XATMI) 269
Datenpuffer
XATMI 256
Datensicherheit 79
Datenstruktur
Name (XATMI) 274
Datentypen
XATMI 256
Deallocate-Aufruf 105
deallocate_type 52
setzen 208
Def-Datei mit EXPORT-Anweisungen 70
Default-Einträge 34
DEFAULT-Name 111
definieren 314
eines Client 94
DEFAULT-Server 94
definieren 304, 311
DEFAULT-Service 94
definieren 304, 311
Deferred_Deallocate-Aufruf 107
DES-Schlüssel 84
destination-name 273
Diagnose
PCMX 344
Trägersystem UPIC 337
Diagnoseunterlagen 337
Disable_UTM_UPIC-Aufruf 109

Dokumentation, Wegweiser 15
Dynamische Bibliothek 70

E
EBCDIC 294
EBCDIC nach ASCII Konvertierung (CPI-C) 68, 103
EBCDIC-Code 71
EBCDIC-Konvertierung XATMI 258
Eigener Anwendungsname 54
Eigenschaften des erweiterten Returncodes
  setzen 244
Eingabeparameter (CPI-C) 96
Einrichten Conversation 100
  Standardwerte 147
Einsatz typisierte Puffer 267
Einschalten
  UPIC-Trace 339
Einschritt-Vorgang, Kommunikation mit 72
Empfangen
  Daten 154
  Daten und Formatkennzeichen 166
  Formate 62
  Nachricht 59
  Teilformate 63
  Teilnachricht 59
Empfangsmodus setzen 231
Empfangspuffer 59
Empfangsstatus 151
Enable_UTM_UPIC
  Aufruf 111
  leerer lokaler Anwendungsname 314
Encryption Level
  zugehörige Schlüsselpaare 85
ENCRYPTION-LEVEL 53
encryption_level
  setzen 191
  zu hoch 192
End-Service (XATMI) 249
Endlosschleife 61
Ereignisse
  XATMI 266
Ermitteln Conversation-ID 148
Erweiterte Information
  abfragen 131
Erweiterter Returncode
  abfragen 134
  Eigenschaften setzen 244
Erzeugen
  LCF 275
Euro-Zeichen
  Konvertierung 69
exit 110
Extract_Client_Context 83, 115
Extract_Communication_Protocol 118
Extract_Conversation_Encryption_Level-Aufruf 120
Extract_Conversation_State-Aufruf 123
Extract_Convertion-Aufruf 125
Extract_Cursor_Offset-Aufruf 127
Extract_Partner_LU_Name-Aufruf 129
Extract_Secondary_Information-Aufruf 131
Extract_Secondary_Return_Code 78, 134
Extract_Shutdown_State 139
Extract_Shutdown_Time 141
Extract_Transaction_State 144

F
F-Tasten
  setzen 210
  Überblick 66
Fehlerbehandlung
  XATMI 266
Fehlerdiagnose
  CPI-C 334
  XATMI 283
Fehlerfall, Verhalten im (CPI-C) 333
Fehlermeldung (CPI-C) 334
Fehlersituation im CPI-C-Programm 110
Ferne Portnummer 295
Filedeskriptor
  Unix-System 328
  Windows 323
Folgezustände einer Conversation 50
FORMAT 65
Format der Namen 294

Formate
  austauschen 62
  empfangen 62
  senden 62
Formatkennzeichen 62
  empfangen 166
  leeres empfangen 174
  senden 181
Formatname 65
Formattyp 65
Fortsetzungszeichen
  LCF 270
FPUT 332
ftp 335
function_key (Set_Function_Key) 210
Funktionen für den Nachrichtenaustausch 56
Funktionsaufrufe 42
Funktionsdeklaration
  Extract_Partner_LU_Name 130
Funktionstasten
  auslösen 66, 210

G
Gemeinsame Datentypen 256
Generierung der UTM-Partner-Anwendung 291
Generierungsbeispiel für Client-Anbindung 349
Grace-Sign-On
  Passwort abgelaufen 80
Größe
  UPIC-Tracedatei 340
Große Datenmengen (XATMI) 269

H
HD 298
Helper Classes 33, 35
HOSTNAME 53
Hostname
  Partner-Anwendung 213
HP-UX 14

I
Identifikation
  einer Conversation 148
Include-Dateien
  CPI-C (Unix-System) 327
  CPI-C (Windows) 321
Initialisieren
  XATMI-Client 261
Initialisierungsparameter
  XATMI 279
Initialisierungswert
  Conversation Characteristics 52, 53
Initialize-Zustand 50, 149
Initialize_Conversation-Aufruf 147
INPUT-Exit 332
int Call 44
int Rcv 42
int RcvMulti 43
int Restart 45
int SetTselFormat 36, 38
int Snd 42
int SndLast 42
int SndRcv 44
Intermediate-Service (XATMI) 249
internal error 334
internal-service-name 272
Interprozesskommunikation 293
IP-ADDRESS 53
IP-Adresse
  Partner-Anwendung 215
IPv4 216, 301, 308
IPv6 217, 301, 308

J
Jobvariable 320
  setzen (Beispiel) 331

K
K&R-Compiler 356
K-Tasten
  setzen 210
  Überblick 66
KCMF 183
kcsaeea.c 70
kcxaent.c 70
KDCDISP 54, 81
  Fehler 157, 169

KDCRECVR (XATMI) 252
    Recovery-Service 272
Kennzeichen
    automatische Konvertierung 298
    Knoten-Anwendung 306
Kerberos
    Returncode 135
    Returncode (Receive) 159
    Returncode (Receive_Mapped_Data) 171
Keycode 79
Klasse CUPIC 26
Knoten 32
Knoten-Anwendung 32
Kommentarzeile
    LCF 270
    upicfile 297
Kommunikation
    mit einem Einschritt-Vorgang 72
    mit einem UTM-Server 71
    mit Mehrschritt-Vorgang 75
Kommunikationsmodell
    asynchron 254
    conversational 255
    synchron 253
Kommunikationsprotokoll
    festlegen in upicfile 302
Kommunikationssystem 31
Kompatibles CPI-C-Programm 235, 355
Komplette Nachricht empfangen 162, 174
Konfigurieren
    mit TNS-Einträgen 296
    ohne upicfile 292
    UPIC mit C++-Klassen 33
    UPIC-L 293
    UPIC-Local mit Visual C++ 324
    UPIC-R 294
    XATMI 270
Konfigurierung des Trägersystems UPIC 291
Konvertierung 68
    ASCII nach EBCDIC (CPI-C) 104
    automatische (CPI-C) 298
    EBCDIC nach ASCII (CPI-C) 103
    Euro-Zeichen 69
    XATMI 258
Konvertierungstabelle 70
    bearbeiten 69
    Windows 103, 104

L

Länge
    Daten (Send_Data) 178
    Daten (Send_Mapped_Data) 181
    der zu sendenden Daten 178
    Formatkennzeichen 166
    Partnername in upicfile 299, 306
    Symbolic Destination Name 96, 299, 306
Langzeitspeicher 79
LC-Definition File 275
LC-Description File 275
LCF 251
    erzeugen 275
Leerer lokaler Anwendungsname 314
Leerer Symbolic Destination Name 293
Leeres Formatkennzeichen empfangen 174
Letzte Ausgabenachricht (XATMI) 252, 272
Linux siehe Unix-Systeme
Linux-Distribution 14
Little Endian 256
LN 312
LN.DEFAULT-Eintrag 111
local client name 261
Local Configuration
    Code für Datentypen 257
Local Configuration Definition File 270
Local Configuration File 251
    erzeugen 275
Local Definition File
    für XATMI-Beispiel-Anwendung 347
local_name 54
    Disable_UTM_UPIC 109
    Enable_UTM_UPIC 111
Lockcode 79
Logging-Datei
    BS2000 338
Logging-Datei (UPIC) 337
Lokale Anbindung (UPIC-L) 29

Lokale Anwendung
  Port setzen 238
  Transport-Selektor setzen 240
  Transport-Selektor-Format setzen 242
Lokale Portnummer 294
Lokaler Anwendungsname 34, 297
  Enable_UTM_UPIC 111, 314
  leer 314
  upicfile 280, 312
Lokaler Sendepuffer 58
LTERM-Pool
  für Mehrfachanmeldungen 95

M
Main-Thread 89
map_name
  Receive_Mapped_Data 166
  Send_Mapped_Data 181
  Überblick 63
map_name_length
  Receive_Mapped_Data 166
  Send_Mapped_Data 181
Maschinenabhängigkeiten 256
Mathemathische Bibliothek 282
mathlib 282
Maximale Nachrichtenlänge 269
Mehrere Conversations 89
Mehrere CPI-C-Programmläufe 113
Mehrfaches Anmelden
  bei UTM 113
  LTERM-Pool für 95
  mit demselben Namen 95, 314
  unter einer Benutzerkennung 78, 80
Mehrschritt-Vorgang 61
  Kommunikation mit 75
Meldungen
  xatmigen 287
Metasyntax 23
MGET 66, 183
  19Z 211
MGET NT 63
Microsoft Visual C++ Developer Studio 70
Migration, CPI-C Version 2 354
MODE
  Service Modell 273
MPUT 74
MPUT NT 59, 61
MSCF 377
Multiple Conversation 89
  Beispiel 92
Multithreading 31, 89

N
Nachricht
  empfangen 59
  senden 57
Nachrichtenaustausch, Funktionen für 56
Nachrichtenlänge
  maximale 269
Name
  XATMI-Datenstruktur 274
Name des Programms
  bei Enable_UTM_UPIC 111
Netzadressierung 296
Neue Conversation 41
Neue local Adresse 40
Neue remote Adresse 40
Neue security Attribute 40
Nicht-blockierender Receive
  Receive 154
  Receive_Mapped_Data 166
  setzen 231
  Überblick 61

O
OCTET STRING 257
Offset
  Cursor-Position 127
  Cursor-Position im Format 68
OpenCPIC Trägersystem 27
openUTM 293
  Benutzerdaten verschlüsseln 84
  Benutzerkonzept 77
  Formatkennzeichen 65
  Funktionstasten 66
openUTM-Anwendung beendet 333
openUTM-Anwendung, siehe UTM-Anwendung

openUTM-Client-Interface 33
openUTM-Cluster
Regeln für upicfile 305
Symbolic Destination Name 297, 305
openUTM-Server 292
Kommunikation mit 71
openUTM-Version
ab V4.0 302
Overhead
Puffer 269

P
Parallele Conversations 89
Partner-Anwendung
Hostname setzen 213
IP-Adresse setzen 215
Port setzen 221
Transport-Selektor-Format setzen 225
Transportselektor setzen 223
partner_LU_name 54, 129, 147, 292
BS2000 299
partner_LU_name_length 129, 147
Partnerkonfiguration
Abstimmung 315
Partnername
in upicfile 299, 306
Passwort
fehlerhaft 199
setzen 198
ungültig 101, 156, 168, 199, 204
PATH 324
PCMX 30, 296, 327
Diagnose 344
PEND ER/FR 335
PEND FI 59, 61
PEND KP 61
PEND RS 332
Pfadname 340
PGPOOL 158, 175, 179, 183
ping 335
PORT 53
Portierbarkeit von CPI-C-Programmen 235
Portnummer 102 294
Prepare_To_Receive
Aufruf 151
Überblick 58
Programm
abmelden (CPI-C) 109
Programmaufrufe (CPI-C) 96
Programmbeispiel
Side Information-Datei (Windows) 346
tpcall (Windows) 347
uptac (Windows) 346
Programmname
angeben (CPI-C) 111
Programmschnittstelle
CPI-C 96
XATMI 259
Property Handlers 40
PROTOCOL 302
abfragen 118
setzen im Programm 189
PTERM-Name
UPIC-Local 112
Public Diagnosefunktion 46
Puffer
definieren 274
erweiterte Information 131
für Daten 57
maximale Größe 269
Sendedaten 179, 183

Q
Quick Start Kit 324

R
Readme-Dateien 20
Receive
Aufruf 154
blockierend 60
mehrere Aufrufe 74
nicht-blockierend 61
Überblick 58, 59
Receive Timer 228
Receive-Zustand 50

Receive_Mapped_Data
  Aufruf 166
  Überblick 58, 59, 63
receive_timer 228
receive_type 52
received_length
  Extract_Shutdown_Time 142
  Receive 155
  Receive_Mapped_Data 167
Rechner-übergreifende Kommunikation 292
Recovery-Service (XATMI) 252
Red Hat 14
Reihenfolge
  CPI-C-Aufrufe 55
Remote Anbindung 30
remote-service-name 272
Request 249
Request-Response
  konfigurieren 273
Requester 250
Reset-Zustand 50
  nach Receive 61
Resource-Dateien 70
RESTART 78, 81
RET 211
return_code 129
return_control 51
Returncodes
  CPI-C 96
Returnwert (XATMI) 266
revent 266
RSA-KEY 53
RSA-Schlüssel 84, 85

S
Schlüsselwörter 54
Schnittstellen 26
SD 298
secondary information 131
Security-Funktionen 78
Security-Typ setzen 201
security_password 52, 198
security_password_length 52
security_type 52
  Fehler bei UTM 101
security_user_ID 52
security_user_ID_length 52
Send-Zustand 50
Send_Data
  Aufruf 178
  mehrere Aufrufe 74
  Überblick 57
send_length
  Send_Data 178
  Send_Mapped_Data 181
Send_Mapped_Data
  Aufruf 181
  Überblick 57, 63
send_type 51
Senden
  Daten 178
  Daten mit Formatkennzeichen 181
  Formate 62
  Nachricht 57
  Teilformat 63
Sendepuffer
  lokal 58
  übertragen an Server 58
Senderecht 50, 56
  anzeigen (Receive) 155
  anzeigen (Receive_Mapped_Data) 167
  empfangen 162, 174
  übergeben 58
Senderichtung
  ändern (Receive) 162
  ändern (Receive_Mapped_Data) 174
Sendestatus 151
Server 25
  XATMI 249
Service
  definieren 271
  XATMI 249
Set-Aufrufe 96
Set_Allocate_Timer-Aufruf 184
Set_Client_Context 83
  Aufruf 186
Set_Communication_Protocol 189

Set_Conversation_Encryption_Level-Aufruf 191
Set_Conversation_Security_New_Password 195
Set_Conversation_Security_Password
  Aufruf 198
  Überblick 78, 80
Set_Conversation_Security_Type
  Aufruf 201
  Überblick 78
Set_Conversation_Security_User_ID
  Aufruf 203
  Überblick 78
Set_Convertion 206
Set_Deallocate_Type-Aufruf 208
Set_Function_Key
  Aufruf 210
  Überblick 66
Set_Partner_Host_Name 295
  Aufruf 213
Set_Partner_IP_Address 295
  Aufruf 215
Set_Partner_LU_Name 218
Set_Partner_Port 295
  Aufruf 221
Set_Partner_Tsel 295
  Aufruf 223
Set_Partner_Tsel_Format 295
  Aufruf 225
Set_Receive_Timer-Aufruf 228
Set_Receive_Type
  Aufruf 231
  Überblick 60
Set_Sync_Level-Aufruf 234
Set_TP_Name-Aufruf 236
Setzen
  deallocate_type 208
  Empfangsmodus 231
  Hostname der Partner-Anwendung 213
  IP-Adresse der Partner-Anwendung 215
  partner_LU_name 218
  Passwort 198
  Port für Partner-Anwendung 221
  Security-Typ 201
  Synchronisationsstufe 234
  Timeout-Timer 228
  TP_name 236
  TP_name_length 236
  Transport-Selektor für lokale Anwendung 240
  Transport-Selektor für Partner-Anwendung 223
  Transport-Selektor-Format für lokale Anwendung 242
  Transport-Selektor-Format für Partner-Anwendung 225
  UTM-Benutzerkennung 203
  UTM-Funktionstaste 210
  Verschlüsselungsebene 191
SFUNC
  RET 211
Shared Memory für UPIC-Local (Unix-System) 329
Shellvariable UPICLOG 338
SHUTDOWN GRACE 139
SHUTDOWN WARN 139
Shutdown-Status
  abfragen 139
Shutdown-Time
  abfragen 141
Side Information 50
Side Information-Datei 50, 297
  für Programmbeispiele (Windows) 346
SIGHUP 328
SIGINT 328
Signale 328
SIGNON-Vorgang 77
SIGQUIT 328
Socket-Schnittstelle 30
Solaris 14
Specify_Local_Port 295
  Aufruf 238
Specify_Local_Tsel 295
  Aufruf 240
Specify_Local_Tsel_Format 295
  Aufruf 242
Specify_Secondary_Return_Code
  setzen 244
stand-alone UTM-Anwendung 11

Start-Zustand 50
Starten
    BS2000 330
    CPI-C-Programm (Unix-System) 328
    CPI-C-Programm (Windows) 321
    Teilprogramm 58
    Thread 89
    UTM-Service 57
Starter-Set 97
status_received
    Receive 155
    Receive_Mapped_Data 167
Struktur
    CPI-C-Anwendung 55
Subtypen (XATMI) 257
SUSE 14
SVCU 271
sym_dest_name (Initialize_Conversation) 148
Symbolic Destination Name 54, 293, 297, 298
    Cluster 305
    Länge 96, 299, 306
    openUTM-Cluster 305
Symbolic_Destination_Name 34
Synchrones Request-Response-Modell 253
Synchronisationsstufe setzen (CPI-C) 234
Syntax
    LCF-Definitionsdatei 270
SYSTEM PEND ER 335

T

T-SEL 53
T-SEL-FORMAT 53
T.61 string 258
T.61-Zeichensatz 358
TAC
    Verschlüsselung 87
TCP/IP 294
    Verbindung überprüfen 335
TCP/IP-Port
    lokale Anwendung 238
    Partner-Anwendung 221
Teilformat
    senden/empfangen 63
Teilnachricht
    empfangen 59
Teilprogramm starten 58
telnet 335
TEMP 337, 340
Thread 89
    beenden 90
    starten 89
Threadfähige Systeme 34
Threading 327
TIMEOUT 246
Timeout 184
    blockierender Receive 158, 170
Timeout-Timer
    setzen für blockierenden Receive 228
    Überblick 61
Timer der UTM-Anwendung 61
Timer für blockierenden Receive 228
TMP 337, 340
TNS-Directory 296
TNS-Eintrag 296
    abstimmen 315
    für Programmbeispiele (Windows) 346
    Windows 346
TNS-Name 111
TP_name 52, 54
    setzen 236
TP_name_length 52
    setzen 236
tpacall 254
tpcall 253
    Programmbeispiel für Windows 347
tpcall.ldf, Local Definition File 347
tpconnect 255
tpdiscon 255
TPEEVENT 266
tperrno 266
tpgetrply 254
tpinit 260, 261
    lokaler Anwendungsname 280
TPOOL 95
tprecv 255
tpsend 255
tpterm 260, 263

Trace
  UPIC 338
Tracedatei
  Größe bei UPIC 340
  Größe bei XATMI 286
  Name bei UPIC 341
  Name bei XATMI 286
  UPIC 341
  Verzeichnis 340
  XATMI 286
Trägersystem
  Anschluss an das (XATMI) 260
  OpenCPIC 27
  UPIC 27
Transaktionscode 54
  in upicfile 300, 306
  setzen im Programm 236
  setzen in C++ Klasse CUpic 41
  ungültig 101, 157
  XATMI 272
Transaktionsstatus 77
  lesen 144
TRANSDATA 294
Transport-Selektor 294
  Format für lokale Anwendung 242
  Format für Partner-Anwendung 225
  lokale Anwendung 240
  Partner-Anwendung 223
Transportsysteme 294
Transportverbindung, Fehler bei Aufbau 100
Typed Buffer 250
Typen (XATMI) 256
Typisierte Puffer 250, 256, 267
  Regeln 267
  Typen 256
Typisierte Records 256

U

Übergeben Senderecht 58
Überprüfen
  TCP/IP-Verbindung 335
Übersetzen
  BS2000 330
  CPI-C-Programm (Unix-System) 327
  CPI-C-Programm (Windows) 321
  XATMI-Programm 282
ULS 79
Umgebungsvariable
  CPI-C 319
  setzen für CPI-C (Windows) 322
  UPIC-Local (Unix-System) 329
  UPICPATH 297
  XATMI 283
Ungültig
  Benutzerkennung 156, 168, 204
  Passwort 101, 156, 168, 199, 204
Unix-Plattform 14
Unix-Systeme 13
Untypisierter Datenstrom 256
UPIC Conversations 34
UPIC Trägersystem 27
UPIC(Windows) Programmbeispiele 345
UPIC-Client 292
upic-cob 347
UPIC-Conversation 26
UPIC-Generierung 297
  XATMI 278
UPIC-Kommunikationsprotokoll
  festlegen in upicfile 302
UPIC-L 31, 293
UPIC-Local 29, 112
  binden bei XATMI 282
  Enable_UTM_UPIC 112
  nicht-blockierender Receive 61, 154
  Partnername 299
  Set_Receive_Timer 228
  Set_Receive_Type 231
  Unix-System 329
  UTMPATH 113
  Windows 324
UPIC-Logging-Datei 334, 337
UPIC-Puffer 57
UPIC-R 294
  mit CMX 296
  Windows 31
UPIC-Remote 30
  binden bei XATMI 283
  Partnername 299

UPIC-spezifisch
  Conversation Characteristics 53
  Funktionen 99
UPIC-Trace 338
  aufbereiten 343
  ausschalten 343
  BS2000 339
  einschalten 339
UPIC.H
  BS2000 330
  Unix-System 327
  Windows 321
UPIC.INI 322, 337
UpicB.ocx 21
UPICFIL 297, 320
UPICFILE 319
upicfile 297
  Enable_UTM_UPIC 111
  lokaler Anwendungsname 280
  XATMI 278
upicfile-Eintrag
  lokale Anwendung 312
  stand-alone UTM-Anwendung 298
  UTM-Cluster-Anwendung 305
UPICL 292
UPICLOG 319, 320, 337, 338
UPICPATH 297, 319
UPICR 292
UpicSimpleClient 348
UPICTRA 320
UPICTRACE 319, 338
upicw32.lib 321
upicws32.lib 321
uptac
  Programmbeispiel für Windows 346
USER 279
Userbuffer 256
UTM 293
UTM-Anwendung beendet 333
UTM-Benutzerkennung
  Mehrfachanmeldung 78
  nicht generiert 101
  setzen 203
  ungültig 156, 168
UTM-Benutzerkonzept 77
UTM-Cluster-Anwendung 11, 32
  Cluster-Administrations-Journal 375
UTM-Fehlermeldung 337
UTM-Formate
  empfangen 166
  senden 181
UTM-Funktionstaste 66
  setzen 210
UTM-Server 292
  Kommunikation mit 71
UTM-Service
  adressieren 54
  starten 57
UTM-Version
  ab V4.0, UPIC-Protokoll für 302
  kleiner V4.0, UPIC-Protokoll für 302
UTM.local 293
utmcnv32.def 70
utmcnv32.rc, resource.h 70
UTMPATH 149
  nicht gesetzt 113
  UPIC-Local (Unix-System) 329

V

Verbindungsabbau
  beim Empfangen 157, 169
  beim Senden 179, 182
  beim Wechseln des Empfangsstatus 152
  durch Beenden von CPI-C 333
Verbindungsaufbau
  bei Verschlüsselung 86
  Probleme beim 335
Verfolgerinformation 338
Verschlüsselung 40, 84
Verschlüsselung am TAC 87
Verschlüsselungsebene 86
  abfragen 120
  ändern 122
  setzen 191
  zugehörige Schlüsselpaare 85
Verschlüsselungsverfahren 84
Verteilte Transaktionsverarbeitung 58, 76
Verwendung von TNS-Einträgen 315

Verzeichnis der Tracedateien 340
Visual C++
  Konfigurieren mit Hilfe von 324
void GetMapName 41
void GetTPName 41
void Reset 46
void SetEncryption 40
void SetFunctionKey 41
void SetHost 38
void SetLocal 40
void SetMapName 41
void SetPort 36, 38
void SetRemote 40
void SetSecurity 40
void SetTPName 41
void SetTselName 36, 38
Voraussetzungen
  Verschlüsselung 85
Voreinstellung
  GLOBALER NAME 315
Vorgangs-Wiederanlauf 78, 81
Vorgangskontext 81
Vorgangsstatus
  lesen 144

W
Wartezeit
  maximale (Receive) 174
Wechseln
  Senderecht 58
  vom Sende- in Empfangsstatus 151
Wiederanlauf 81
  Client-Kontext 83
  XATMI 252
Wiederanlauf-Programm (CPI-C) 81
Windows, UPIC-Local 324
Windows-System 14
WINDOWS.H 321
Worker Thread 34
Wrapper class 33
Wrapper Class CUpic 26

X
X/Open CPI-C-Schnittstelle 353
X_C_TYPE 256, 257
  Konvertierung 258
X_COMMON 256, 257
  Konvertierung 258
X_OCTET 256
  BUFFERS-Operand 273
XATMI
  Anwendung 27
  Client abmelden 263
  Programmschnittstelle 259
  Standard 249
  U-ASE 251, 256
XATMI-Programm 27
  Zugriff auf Services 28
xatmigen 275
  aufrufen 276
XTLCF 284
XTPALCF 284
XTPATH 283
XTSVRTR 284

Z
Zeichensatzcodierung 258
Zeichensatzkonvertierung
  automatische (CPI-C) 298
  CPI-C 68
  XATMI 258
Zeilenabschlusszeichen, upicfile 303, 309
Zugangsinformationen, Umfang bestimmen 201
Zugangsprüfung
  ohne Passwort 198, 202
  UTM 279
Zugangsschutzfunktionen 78
Zugriffsschutzfunktionen 79
Zurücksetzen
  Receive-Timer 228
Zusätzliche Funktion des UPIC-
  Trägersystems 184
Zustand
  "Initialize" 50, 149
  "Receive" 50
  "Receive", wechseln in 151
  "Reset" 50
  "Send" 50, 162

Zustand (Forts.)
"Send", ändern in "Receive" 151
"Start" 50
Conversation 50
der Conversation abfragen 123
Zustandstabelle 359
Zweistufig
partner_LU_name 292, 294
Partnername 299, 306